

**DATABASES**

**DATABASES**

**DATABASES**

**DATABASES**

**SIMPLIFIED USER'S GUIDE**

**For Cassette, Diskette, and Disk Systems**

**DATAPOINT CORPORATION** 

# **DATABUS**

## **Simplified User's Guide**

The text in this manual was entered, edited, and typeset using a Datapoint 2200 with a phototypesetter and the Scribe Text Processing Program.

**Manual No. 50004**

**Revised February 1976**

Copyright©1976 by Datapoint Corporation  
Printed in U.S.A.

The "D" logo, Datapoint, Datashare, Dataform, Databus, Datapoll, Scribe, and The Leader in Dispersed Data Processing are trademarks of Datapoint Corporation. Registered in the U.S. Patent Office.

## **FORWARD**

### **A Universal Language**

Datapoint processors are scattered throughout the world, meeting the needs of thousands of users. Each of these applications is unique, yet one key element can tie them together -- the DATABUS language. DATABUS is one of a variety of computing languages, including DATAFORM, RPG, and BASIC, available for use on Datapoint systems. DATABUS is a total business programming language that lets you fully utilize all of the capabilities of your particular Datapoint computing system. Whether you are using a Cassette 1100, Diskette 1100, 2200 processor, or a 5500 processor, DATABUS is the computing language that can solve many of your computing problems.

Because DATABUS is a high-level language, many of the detailed programming aspects of computer usage are taken care of by the DATABUS compiler, leaving you to code your programming application in simple, English-like statements. DATABUS enjoys a special status in the Datapoint family of computer languages because it constitutes a powerful and sophisticated tool in the hands of a professional programmer and at the same time permits easy mastery and use by the non-professional programmer, such as a business executive, for applications essential to his job duties.

DATABUS is not only one of several languages available for single-user computing applications, but it also is the computing language used in all DATASHARE applications. DATASHARE is Datapoint's business timesharing system that can enable up to 8 or 16 users to simultaneously access the computing capabilities of a single 2200 or 5500 processor.

This book is written primarily for the once-in-a-while programmers who find it convenient and useful to be able to solve computing problems with an easy-to-use small business computer, such as a Datapoint 2200. However, it also provides a quick introduction to the language for experienced programmers. The study of this text should, in a short time, allow you to create and operate your own programs on a Datapoint system and will, at the same time, give you a much better "feel" for your company's total computing operation. Computers are, of course, such an integral part of today's business world that it is difficult to function effectively as an executive without knowledge of how they operate, their potential, and their limitations.

## Table of Contents

CHAPTER 1	How to Use This Book .....	1
	A Word About DATABUS .....	3
	About Business Programming .....	3
	Programs and Files .....	3
	Programming is Only Half the Job .....	4
	The Business Organization .....	4
	People are the Critical Ingredient .....	4
CHAPTER 2	Flow Charting .....	5
CHAPTER 3	DATABUS Lesson 1 - Displaying on the Screen .....	11
	The DISPLAY Instruction .....	13
	Cursor Positioning .....	15
	A Few More Examples .....	15
	Chapter Summary .....	18
	Problems .....	18
CHAPTER 4	DATABUS Lesson 2 - Using the Keyboard for Data Entry .....	19
	The KEYIN Instruction .....	21
	Label Rules .....	23
	"Initialized" Labels .....	24
	More on KEYIN .....	25
	Chapter Summary .....	26
	Problems .....	27
CHAPTER 5	DATABUS Lesson 3 - Printing the Data .....	29
	The PRINT Instruction .....	31
	A Printing Program .....	32
	Chapter Summary .....	34
	Problems .....	34
CHAPTER 6	DATABUS Lesson 4 - Arithmetic .....	35
	FORM Reserves Space for Numbers .....	37
	The Arithmetic Operations .....	37
	Constants Can Also be Used .....	42
	Here are Some Precautions .....	42
	Chapter Summary .....	43
	Problem .....	43
CHAPTER 7	DATABUS Lesson 5 - Loops .....	45
	Conditional Loops .....	48
	COMPARE is Used with Numerics .....	49
	MATCH is Used with Characters .....	50
	Beeping .....	52
	Chapter Summary .....	53
	Problems .....	53
CHAPTER 8	DATABUS Lesson 6 - Disk Operations .....	55
	Each Disk Contains Files .....	57
	A File is a Collection of Records .....	57
	It's Easy to Read From and Write to Disk .....	58
	A Program to Read Data From Disk .....	61
	A Program that Writes and Reads .....	63
	Adding Data to the File .....	65
	Chapter Summary .....	67
	Problem .....	68

CHAPTER 9	DATABUS Lesson 7 - Using Subroutines .....	69
	Linking Up with other DATABUS Programs .....	72
	Common Data Areas.....	72
	Chapter Summary.....	73
CHAPTER 10	Running Your DATABUS Program.....	75
	Step 1 - The DBCMP Program.....	77
	If You've Got a Diskette System.....	77
	If You've Got a Cartridge or Mass Storage Disk System .....	79
	Step 2 - Type in the Program .....	80
	Step 3 - Compile Your Program Error Messages.....	82
	Step 4 - Run Your Program.....	83
	Step 5 - Fix Your Program.....	84
	General Hints .....	84
CHAPTER 11	For DATASHARE Users .....	85
	What is DATASHARE?.....	87
	The DATABUS Compiler .....	87
	The DATASHARE Interpreter .....	87
	Put the Programs on Your Disk.....	88
	Set Up ANSWER AND MASTER Programs.....	89
	The ANSWER Program .....	90
	The MASTER Program .....	91
	The DSCON Program .....	92
	DATASHARE System Operation.....	94
	Taking Down the System.....	95
CHAPTER 12	DATABUS 2 - For Cassettes .....	97
	Part 1 - Language Differences.....	99
	Label Differences.....	99
	KEYIN and DISPLAY Differences.....	99
	Constant Data Values.....	100
	Data Storage Statements.....	101
	Part 2 - Data Storage on Cassette Tape.....	101
	A Few Guidelines.....	101
	Writing Data to Cassette Tape .....	102
	Reading from Cassettes .....	104
	How Data is Stored on Cassettes.....	105
	Updating the File .....	107
	Part 3 - Running the Program .....	110
	Step 1 - Type In and Edit Your Program.....	110
	Step 2 - Compiling Your Program .....	113
	Compiler Error Messages .....	114
	Step 3 - The Interpreter.....	116
	File Numbers .....	116
	The Interpreter .....	117
APPENDIX A - Instruction Summary .....		119
	Syntactic Definitions .....	119
	DATABUS 11 Language Summary.....	120
	DATABUS 2 Language Summary .....	122
APPENDIX B - Editor Commands .....		125

## **CHAPTER ONE**

### **How to Use This Book**



This book does not cover all of the capabilities of DATABUS, nor is it intended to make you an expert programmer. Only fundamental concepts are covered, though enough insights are given to allow further study by reference to the DATABUS User's Guide. When you are through, you should be able to create a program of your own for any practical business purpose.

Chapters 2 through 9 are basic lessons in DATABUS programming concepts. Chapter 10 deals with actually typing in and editing and compiling a program so the Datapoint processor can accept and run it. It is not necessary to read the entire book, for instance, if all you want is to display a message on the screen. The first lesson will arm you with enough knowledge to do that. Then you simply skip over to Chapter 10 to learn how to type in and compile and run your program.

Chapter 11 is provided for DATASHARE users. It provides you with the basic knowledge so you can set up and start running a DATASHARE system.

Read as much as you need. If for some reason you really get hooked, write for a copy of Advanced Techniques in DATASHARE and the DATABUS Reference Manuals. This Simplified User's Guide is intended to give you a start towards becoming a more confident and competent DATABUS user.

The DATABUS language covered in these chapters is the DATABUS language used in all disk-based systems. Users interested in using only cassettes should check Chapter 12 for the particulars of cassette DATABUS.

## **A Word About DATABUS**

DATABUS is a high-level business-oriented programming language developed by Datapoint. Although similar to other high-level languages such as COBOL, DATABUS takes advantage of the processor architecture to provide simple programming and fast running programs.

To run a DATABUS program, you must first type in the English language style instructions. Then the DATABUS compiler translates these instructions into machine readable code. The last step involves the use of the DATABUS interpreter. The interpreter loads the machine code into the computer's memory and actually executes your program.

While most languages use only a compiler or an interpreter, DATABUS takes advantage of the features of both.

## **About Business Processing**

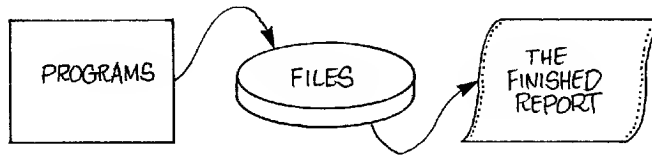
### **Programs and Files**

Business data processing deals primarily with creating files of information, such as a personnel file, and then creating programs that act upon those files to provide other information on a report. An example would be a payroll program that operates in conjunction with the personnel file to produce paychecks. Other programs may use the same personnel file to provide reports indicating current personnel status.

With this in mind, business data processing shouldn't be considered as just programming - it is a combination of data files and programs.



The two, while distinct, aren't separable. The prospective business programmer should view business processing as two aspects rather than simply a program. Many business systems are clumsy in operation due to an inefficient file structure.



## PROGRAMS USE THE CONTENTS OF FILES TO PRODUCE REPORTS

### **Programming is Only Half the Job**

A professional business programmer will probably admit that 50 percent of a workday is spent discovering what management wants done. The other half could be accounted for as actual programming. Why so much non-programming time? A typical request from a company manager might require a summary of all parts selling for over \$50.00 currently stocked in inventory. The programmer will then ask how the print-out format should look, does the manager want all the description of the part, is the quantity on-hand needed and should the report be alphabetically listed or in order of ascending part number?

If several people are involved in the operation, the programmer might have to consult with each. Many managers are accustomed to dealing only with clerks who are familiar with their operations and can usually guess what type of information is needed. The programmer generally doesn't have this load of familiarity and must investigate the requirements thoroughly before beginning any programming task.

### **The Business Organization**

Many business operations are informally organized with each paperwork decision made by a number of administrative or clerical personnel. Many of these organizations have been shaken when a computer attempts to learn the various rules and procedures that weren't well defined in the first place. People, unlike computers, tend to learn their tasks almost by osmosis and can handle the internal politics and exceptions that are inherent to all business paper systems.

### **People are the Critical Ingredient**

As you become an accomplished business programmer, keep in mind that it's still the people that count the most. Attention to the needs of the people who operate and make use of your system will be the factor that makes your system productive or a bust, no matter what the particular features of either the equipment or software may be.

## **CHAPTER TWO**

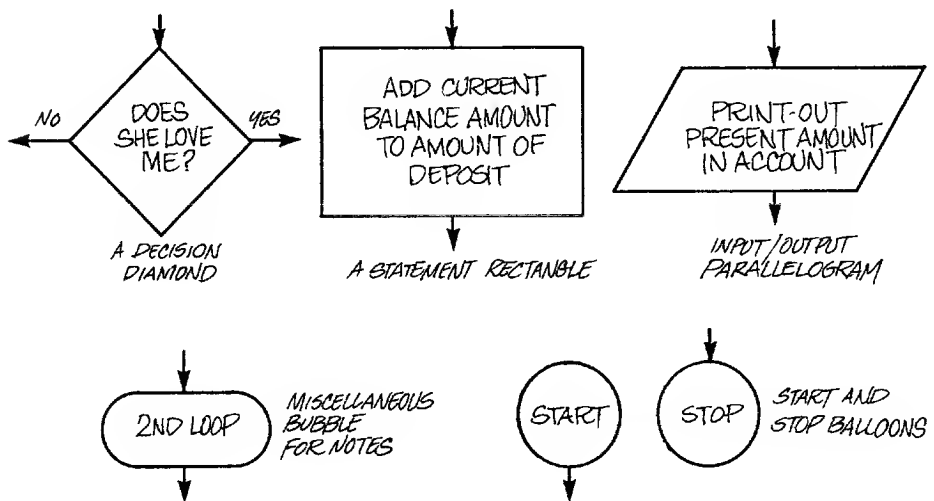
### **Flow Charting**



Flow charting is a method of schematically illustrating a thought process. Flow charting is an essential prerequisite to effective programming. Using the simple flow charting symbols, a programmer can outline and visualize how a program should work. Entire books have been written about flow charting, but we need only deal with the basics.

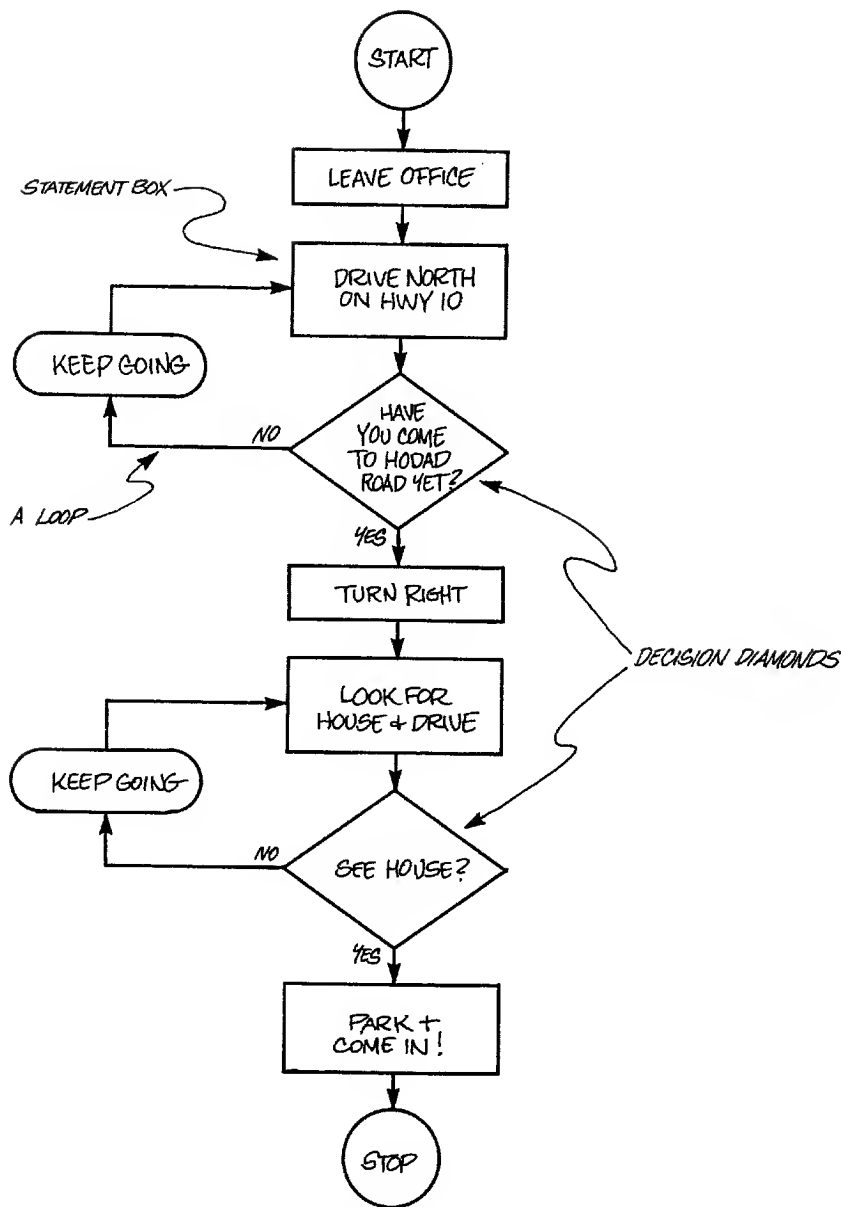
You should be able to define almost any task by using five fundamental flow chart symbols. If your flow chart contains enough detail, your program should almost be a step-for-step textual replica. Simple programs do not always require a flow chart but, at the onset of your programming career, flow chart everything.

## FLOW CHART ESSENTIALS



Using these symbols, you can plot, in a step-by-step fashion, any decision-making process. All possible decisions must be charted along with all of the factors that add up to those decisions. Note that no two things can happen at the same time; instead, everything is done in steps.

The following examples offer some insights into the logical thought processes that make up a flow chart.



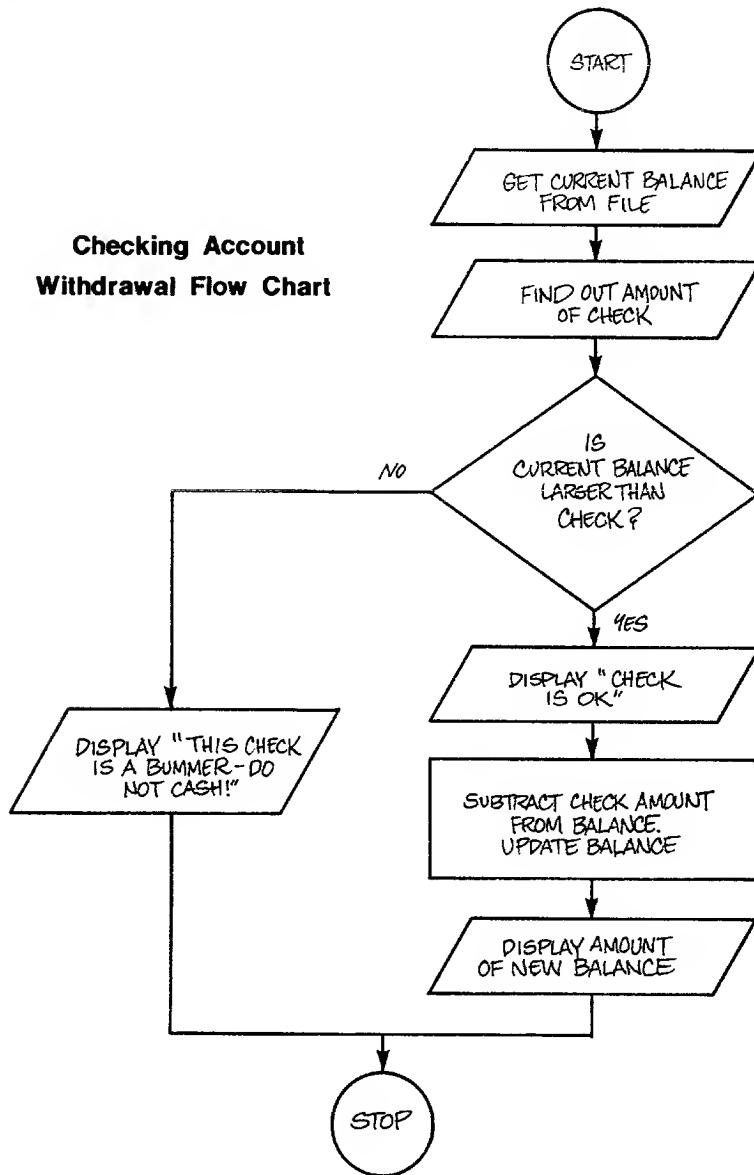
FRED'S FLOW CHART MAP TO HIS HOUSE

Fred could have added more to his flowchart map to cover someone getting lost, or a flat tire, or another calamity, but enough detail has been shown to demonstrate the pains one must take to insure clarity and to assure that events are taken in their logical order.

Note also that we must take one step after another to make sure our plan fits the task at hand. If you're the kind of person who outlined all his school reports before writing them, you'll have no trouble with programming.

**Problem:** Flowchart the process of withdrawing money from your checking account. Check for overdrawing. Use decision diamonds and statement boxes.

**Solution**



## NOTES

## **CHAPTER THREE**

### **DATABUS Lesson 1 Displaying on the Screen**





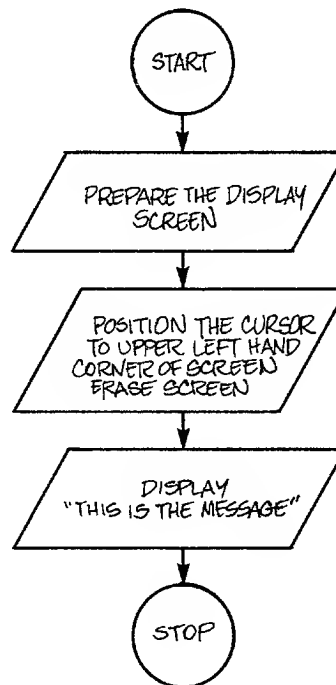
The DISPLAY instruction allows you to display information on the screen of your Datapoint processor. These messages are essential to all DATABUS programs because they provide an easy and effective method for observing the operation of the computer and they will let you know what stage of the program you are in.

### The DISPLAY Instruction

With this in mind, we'll write a short program to put some text on the screen, which is 12 lines deep by 80 columns wide. You can put your message anywhere you like.

Incidentally, you'll see the word "cursor" frequently. It's shorthand for the across-and-down position in which the letters will appear on the screen.

Let's write a short program to put a message on the screen. We can even flow chart it. By the way, as you study this booklet, leave lots of flow charts lying around your desk and on lunch napkins. Before you know it, you'll be considered the staff expert on computers.



Now that we have the flow chart, we'll jump ahead and write the program even before you know the rules.

```
DISPLAY *ES,"THIS IS THE MESSAGE"  
STOP
```

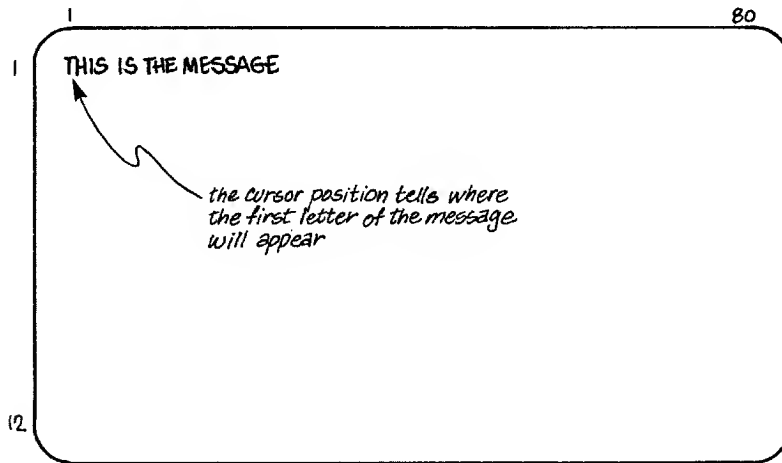
Not too bad, right? Remember that the computer reads the instructions from left to right and executes each operation as it comes to it.

See how obvious DATABUS instructions can be? The cursor position for the message is defined by the \*ES command. \*ES tells the

computer to erase the entire screen and position the cursor at the upper left hand corner. It's a good idea to erase the screen, or at least part of it, before you display a message to wipe out old love letters or whatever was left on the screen when you arrived.

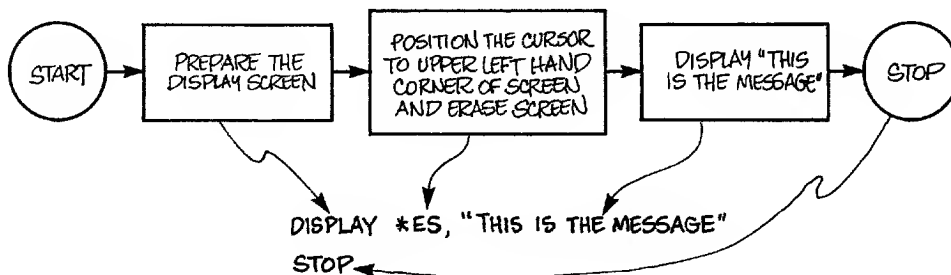
The quotation marks around the message itself tell the computer what is to be displayed. Since the quotes indicate where to start and stop displaying, you shouldn't use them as part of your message.

The following illustration shows what our short program will produce. Notice that once we have defined the beginning cursor position, the rest of the message falls into place after it and no further definition of the cursor position is necessary.



THE DATAPOINT PROCESSOR DISPLAY SCREEN

If this is confusing, let's compare the program to the flow chart. The flow chart remains the same except that it's now written horizontally for easy comparison to the actual program.



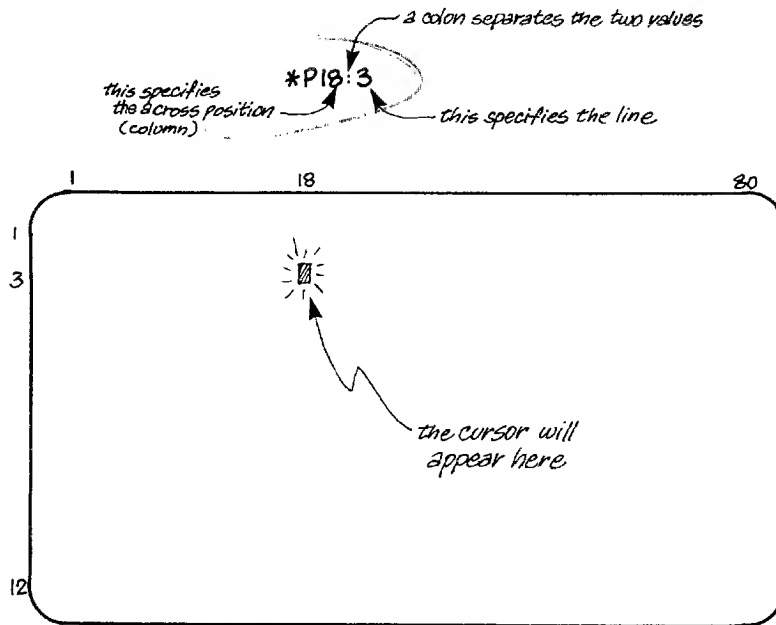
Flow charts can be very helpful, as you can see, if you include enough details. Once you fully grasp the concept, your flow charts will be more abbreviated.

You'll see that the example programs have a STOP instruction at the end. This tells the computer that there are no more instructions and to stop running this program.

## Cursor Positioning

What if you don't want the cursor positioned at the upper left-hand corner of the screen? You can move the cursor to any position on the screen using other cursor positioning controls.

The \*P control positions the cursor to a particular place on the screen. You can specify that position in the following manner:



The \*EF control erases the screen from the current cursor position. So, for example, to erase the bottom half of the screen, use this DISPLAY statement:

*1st column* → *6th row*  
`DISPLAY *P1:6, *EF` ← *Erase the rest of the screen*

The \*EL control erases the rest of the line starting from the current cursor position.

The \*N control positions the cursor at the beginning of the next line.

And, finally, the \*R control makes the screen roll up (the characters in row 12 move up to row 11, the characters in row 11 move up to row 10, etc., and the characters in row 1 are lost).

## A Few More Examples

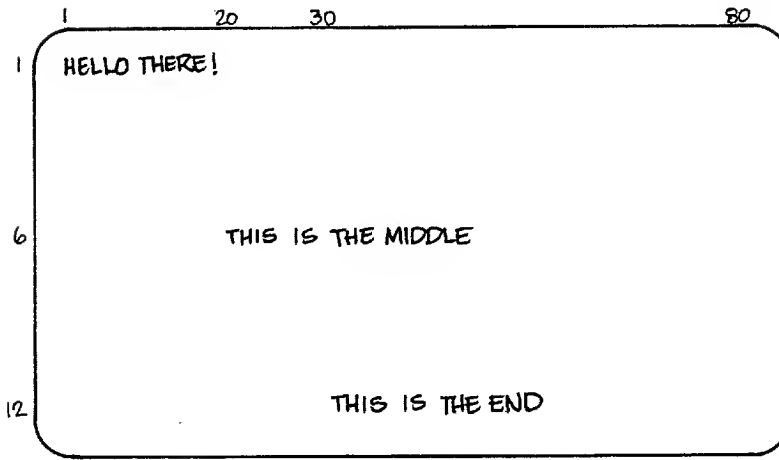
To see how these controls can be used, let's look at some more examples.

Let's write a program to display a more complex message. Look at the program and then look at the result.

```

DISPLAY  *ES,"HELLO THERE!";*P20:6:
        "THIS IS THE MIDDLE";*P30:12:
        "THIS IS THE END"
STOP

```



THE DATAPOINT PROCESSOR SCREEN

Did you notice that there was a colon (:) at the end of the program's first sentence? That colon is a handy little device as it allows you to write instructions that are longer than the space of a single line. In effect, the colon tells the computer to "keep reading".

If it weren't for the colon, we would have to write it this way:

```

DISPLAY  *ES,"HELLO THERE!"
DISPLAY  *P20:6,"THIS IS THE MIDDLE"
DISPLAY  *P30:12,"THIS IS THE END"
STOP

```

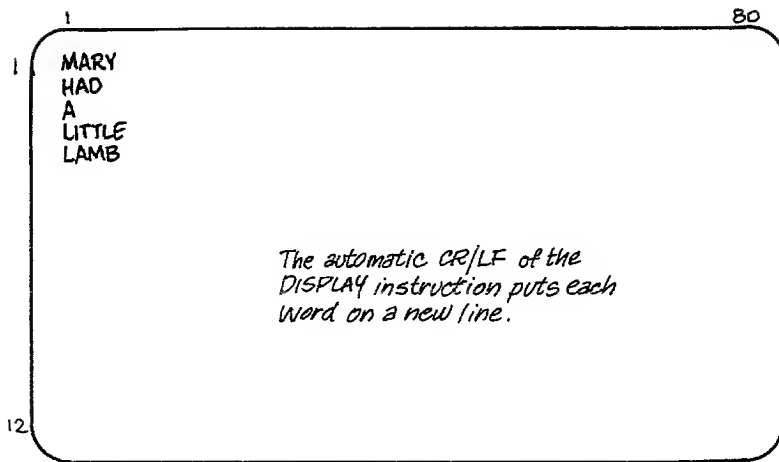
Not only is the short method using the colon faster to write, but it also saves processor time! Use the colon and one lone DISPLAY statement rather than several short DISPLAY statements.

You're almost finished with DISPLAY, but we should cover one more thing. Every time the computer finishes a DISPLAY statement, it automatically sends a carriage return (CR) and line feed (LF) to the screen. The names imply the same operation that occurs when you hit the return key on a regular typewriter. Unless told otherwise, the cursor jumps to the beginning of the next line. This example should help explain things.

```

DISPLAY  *ES,"MARY"
DISPLAY  "HAD"
DISPLAY  "A"
DISPLAY  "LITTLE"
DISPLAY  "LAMB"
STOP

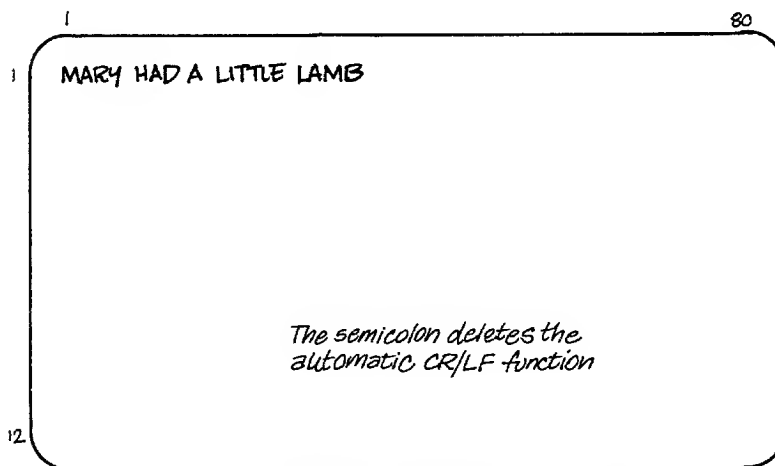
```



THE DATAPOINT PROCESSOR SCREEN

At some time, you might not want this to happen. You might want to leave the cursor where it was for one reason or another. If this is the case, end the DISPLAY instruction with the semicolon (;). This cancels the automatic CR/LF function.

```
DISPLAY *ES,"MARY ";
DISPLAY "HAD ";
DISPLAY "A ";
DISPLAY "LITTLE ";
DISPLAY "LAMB "
STOP
```



THE DATAPOINT PROCESSOR SCREEN

Naturally, we could have written the sentence on the screen originally by putting the entire thing in one program sentence, but you can see how the semicolon works. Notice that each word after DISPLAY has a space after it. This was done so that we wouldn't end up with something like MARYHADALITTLELAMB, which is perhaps economic in space but aesthetically unappealing.

That's it. If all you want to do is get something on the screen, jump to Chapter 10.

## Chapter Summary

DISPLAY	The screen instruction
*Pc:d	The cursor position in column c line d
*ES	Erases screen and positions cursor to upper left corner
*EF	Erases the rest of the screen from the current cursor position
*EL	Erases to end of line
*N	Moves cursor to the beginning of the next line
*R	Moves all lines up one and the top line is rolled off the screen
"MESSAGE"	Message must be bracketed by quotes (programmers call this a literal)
:	Allows some instructions to continue to another line
;	Suppresses automatic carriage return and line feed in DISPLAY instruction
STOP	The end of the program

## Problems

1. Write a program that will erase the screen and then display your name in the center of the screen.
2. Problem 1, but erase it after you write it, then write it again.

## Solutions

1.           DISPLAY   \*ES,\*P33:6,"ATTILA THE HUN"  
              STOP
2.           DISPLAY   \*ES,\*P33:6,"ATTILA THE HUN";  
              DISPLAY   \*P33:6,\*EL,"ATTILA THE HUN"  
              STOP  
  
              or  
  
              DISPLAY   \*ES,\*P33:6,"ATTILA THE HUN"  
              DISPLAY   \*ES,\*P33:6,"ATTILA THE HUN"  
              STOP

## **CHAPTER FOUR**

### **DATABUS Lesson 2 Using the Keyboard for Data Entry**





If you have ever sat in on any Datapoint demonstrations, you know that the keyboard provides one of the primary means of communicating to the processor what it is you wish done. You also know that if the program running in the Datapoint computer is written so as to ignore the keyboard, you can pound on the keys forever and still have no effect on the computer's actions.

It's evident that, like the display screen, the keyboard must be recognized by the computer in order to enter data. The Datapoint's keyboard is almost identical to that of a standard office typewriter with the addition of an adding machine keyboard (numeric pad) and five special function keys. Upper and lower case letters can be used, from the keyboard and to the display screen.

### **The KEYIN Instruction**

With the DISPLAY instruction, we had our messages all neatly tucked between the quotes, and we effectively reserved space for the message when the program was loaded into the computer.

KEYIN - the term for keyboard entered information - is somewhat different. You don't know exactly what an operator might type in as a response to a question. But space must be reserved to accommodate this incoming data. For instance, if the operator was to answer a query regarding a person's name via the keyboard, and that person's name was Harold Joy Hupmobile when the longest name you had planned for was Bill Ford, Harold would lose some of his name because the program would not accept such a long name.

How do we reserve space for incoming data? Simply by staking a claim on some space and putting a name on it. Many programming books use the analogy of the mailman's pigeon hole sorting-case with addresses indicated for each "hole".

That's pretty close, and you should keep the idea of a space with a label on it in mind.

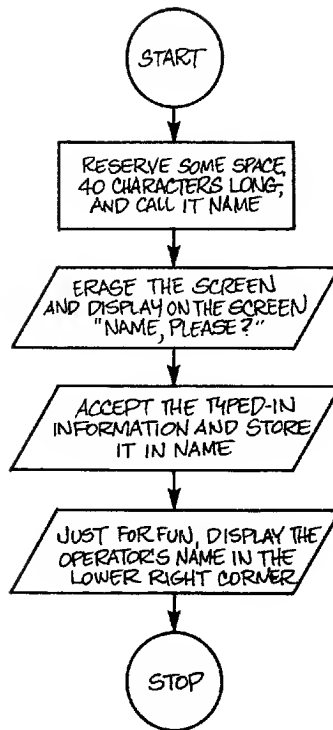
We'll get back to the problem of reserving space after we examine the KEYIN instruction.

KEYIN shares several traits of the DISPLAY instruction so, aside from the new concept of storing the incoming keyboard data, much of this concept will be already familiar to you.

Let's write a short program to ask the operator to type in his name. Since we're still fairly new at this game, a flow chart will help organize the task and will also impress any onlookers that this must be a very erudite and technical booklet.

And, for fun, we'll display back to the operator what his name is, as if he didn't know.

FLOW CHART

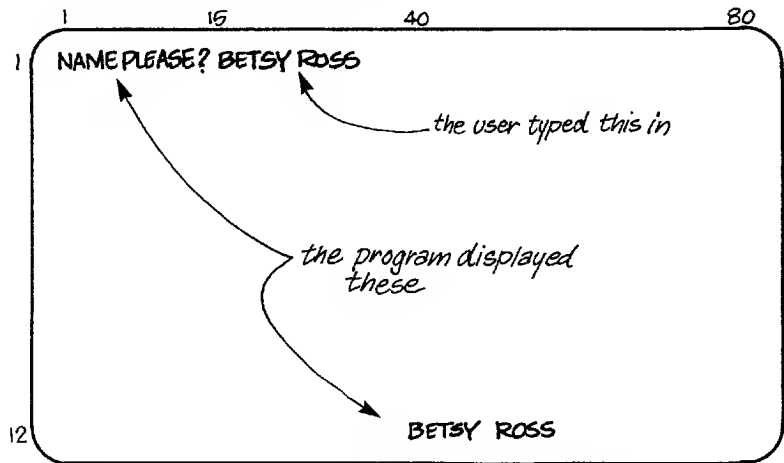


You probably have noticed that this program doesn't accomplish anything useful but it should serve as a good example. Look at the instructions:

*this is a label* → NAME  
*these are instructions* →

DIM	40
DISPLAY	*ES,"NAME, PLEASE?"
KEYIN	*P15:1,NAME
DISPLAY	*P40:12,NAME
STOP	

Now look at the result of the program:



THE DATAPOINT PROCESSOR SCREEN

Let's look at the program and find out what happened. Notice the NAME DIM 40 statement. This looks complex, but the end result is a space labeled NAME with room for 40 characters. You might think of this as "dimensioning" a space in the computer called NAME to allow for a typed input of up to 40 characters.

The DISPLAY instruction should be an old friend by now. Note that we cleared the screen with a \*ES and asked the question.

Look closely at the KEYIN instruction. \*P15:1 should be familiar to you from the DISPLAY instruction. This tells the computer where on the screen the user is to enter the data. But, instead of a message in quotes, we now have the label NAME. Several things happen here and all are important.

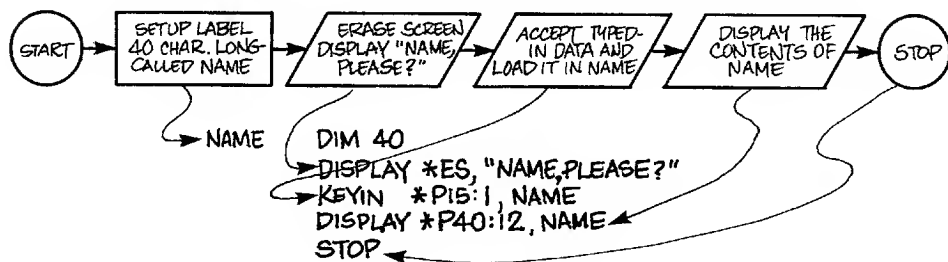
First, the NAME label has been previously defined in the NAME DIM 40 statement. The KEYIN instruction loads the data into that reserved space.

Secondly, when keyboard data is required from the user, the cursor actually is visible as a flashing rectangle beginning at the spot defined by the \*P position coordinates. As the user types in his name, the flashing cursor automatically moves over to indicate the next adjacent character position.

Thirdly, if the operator attempts to type in more than 40 characters, including punctuation and spaces, the cursor will halt in the fortieth space and the Datapoint will emit an audio "beep" to indicate that the limit of the space allowed has been reached.

Lastly, the user tells the computer that he's finished entering the name by tapping the ENTER Key. The flashing cursor will disappear, and the name will be loaded into the label NAME. Up until the time the ENTER Key is pressed, the name can be modified by use of the CANCEL or BACKSPACE Keys.

*HORIZONTAL FLOWCHART AND PROGRAM FOR KEYIN INSTRUCTION EXAMPLE*



## Label Rules

One more thing. Did you notice that the DISPLAY \*P40:12,NAME instruction had no quotes? In this case, we wanted the contents of NAME displayed. No quotes are necessary. They are only needed when we want a specific message displayed. More on this later. As you can see, the use of labels provides considerable power. Here are some guidelines concerning labels or categories of data, since you'll undoubtedly want to make up some labels of your own:

## LABEL RULES

- a. No more than 8 characters
- b. Begin with an alphabetic character
- c. Don't use the same name twice - you'll confuse the computer
- d. All labels must be written at the first part of program

## GOOD LABELS

LOOP  
TESTDATE  
C125

## LABELS THAT WON'T WORK

73SKDO (begins with a number)  
FIRSTNAME (too long)  
!!HELP (illegal first character)

If nothing else, labels can save a pile of work, and you should become familiar with them. We've discussed the DIM or "dimension" labels, and there's one more that's very useful.

```
MESSAGE INIT      "NAME, PLEASE?"
NAME      DIM      40
          DISPLAY   *ES,MESSAGE
          KEYIN     *P15:1,NAME
          DISPLAY   *P40:12,NAME
          STOP
```

### "Initialized" Labels

This program produces exactly the same result as the earlier example. The label MESSAGE has been "initialized" by the INIT statement to contain a specific message. The DISPLAY instruction finds MESSAGE and displays its contents; in this case, the message "NAME, PLEASE?"

At first glance, it appears that there is only a slight difference between the DIM and INIT statements. It is an important difference, however, and bears scrutiny.

If you write NAME DIM 40, the computer saves 40 character spaces, fills them with the equivalent of blanks and attaches to them a label, in this case, NAME. We primarily use this when incoming data is to be stored, such as keyboard inputs.

If we wrote NAME INIT "GEORGE GORDON", the computer reserves a space of 13 characters (12 characters plus the blank space which counts as a character) and places GEORGE GORDON in those spaces. The INIT statement is very handy when you might want to display one message many times during a job. By using INIT and the label, you only have to define it once and then simply call it by its name, which is the label. Data can be loaded into an INIT space just as with a DIM, but once you've done that, the original message or data is lost.

## More On KEYIN

Most programming languages contain an assortment of shortcuts that allow a system programmer to produce more work over a given amount of time. DATABUS is no exception. The following example illustrates a shortcut that can be done by means previously explained, and if a substantial task is planned, this technique might save time.

By this time you're familiar with our enter-your-name program:

```
NAME      DIM      40
          DISPLAY  *ES,"NAME,PLEASE?"
          KEYIN    *P15:1,NAME
          DISPLAY  *P40:12,NAME
          STOP
```

We can write this same program and save a step by using KEYIN's power of display.

```
NAME      DIM      40
          KEYIN    *ES,"NAME,PLEASE? ",NAME
          DISPLAY  *P40:12,NAME
          STOP
```

That's pretty strange, right? But the result is still the same. But now we have eliminated one of the DISPLAY instructions and made KEYIN do the work.

To understand this, simply remember that if you write something between quotes in the KEYIN instruction, it will be displayed, and the cursor will appear immediately after the displayed text and await the user's response.

Why have this feature in KEYIN? Many application programs involve a fill-in-the-form technique, and this helps speed up the process. Here's a short program that asks the name and address of the user.

```
NAME      DIM      20
ADDR      DIM      20
CITY      DIM      20
STATE     DIM      15
          KEYIN    *ES,"NAME: ",NAME
          KEYIN    *P40:1,"ADDRESS: ",ADDR
          KEYIN    *P40:2,"CITY: ",CITY:
                  "STATE: ",STATE
          STOP
```

Aside from the labels, the program uses only one type of instruction. This is confusing but follow through the KEYIN instruction. Notice that it displays and then waits for the operator to keyin the information.

The result appears as this:

1 40 80

1 NAME: GEORGE JONES ADDRESS: 105 ROCK ST.  
CITY: BOULDER STATE: COLORADO

*The KEYIN instructions yield  
an effective data entry program*

12

#### THE DATAPOINT PROCESSOR SCREEN

We used the display power of KEYIN to show the operator the questions. Note that the colon used was inside the quotes - don't confuse it with the colon that allows you to write instructions larger than one sentence. The second and third KEYIN instructions did not start with a positioning control. Why? The auto CR/LF function put the cursor at the beginning of the next line.

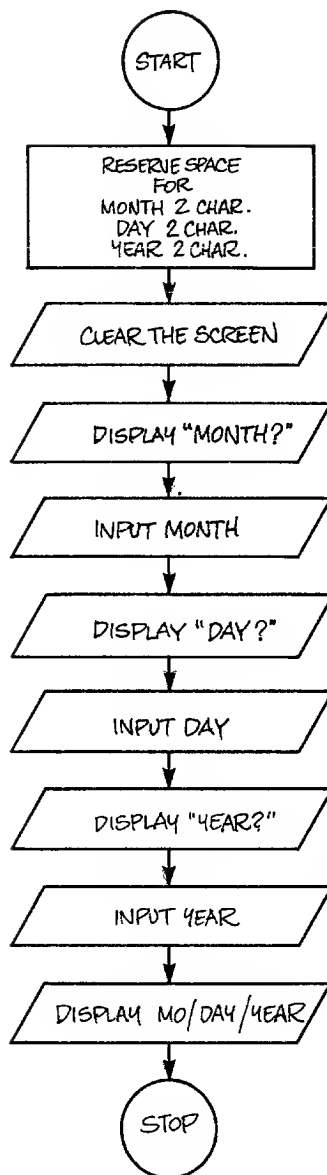
#### Chapter Summary

KEYIN	Accepts data from keyboard and stores it in labeled areas.
*Pc:d	The cursor position in column c of line d.
*ES	Erase screen. (See DISPLAY -- Chapter 3 -- for more features)
label DIM amount	Names and reserves space for data.
label INIT "message"	Allows storage of message and later use by label and name.

#### Problems:

1. Ask for the data to be typed in as answers to three questions, month?, day?, year?. Display the date back in a MO/DAY/YR format.

**Solution:**



**Program:**

MONTH	DIM	2
DAY	DIM	2
YEAR	DIM	2
DISPLAY *ES,"MONTH?"		
KEYIN *P7:1,MONTH		
DISPLAY *P10:1,"DAY?"		
KEYIN *P15:1,DAY		
DISPLAY *P18:1,"YEAR?"		
KEYIN *P24:1,YEAR		
DISPLAY *P1:11, MONTH,"/",DAY,"/",YEAR		
STOP		



2. Write problem number 1 using only one DISPLAY instruction. Also, take advantage of the automatic carriage return and line feed that each KEYIN and DISPLAY instructions gives to save some work.

**Solution:**

```
MONTH   DIM      2
DAY     DIM      2
YEAR    DIM      2
        KEYIN    *ES,"MONTH? ",MONTH,*P10:1:
        "DAY; ",DAY,*P18:1,"YEAR? ",YEAR
        DISPLAY  *P1:11,MONTH,"/",DAY,"/",YEAR
        STOP
```

1 10 18 80

1 MONTH ? 9 DAY ? 14 YEAR ? 76

11 9/14/76

12

THE DATAPOINT PROCESSOR SCREEN

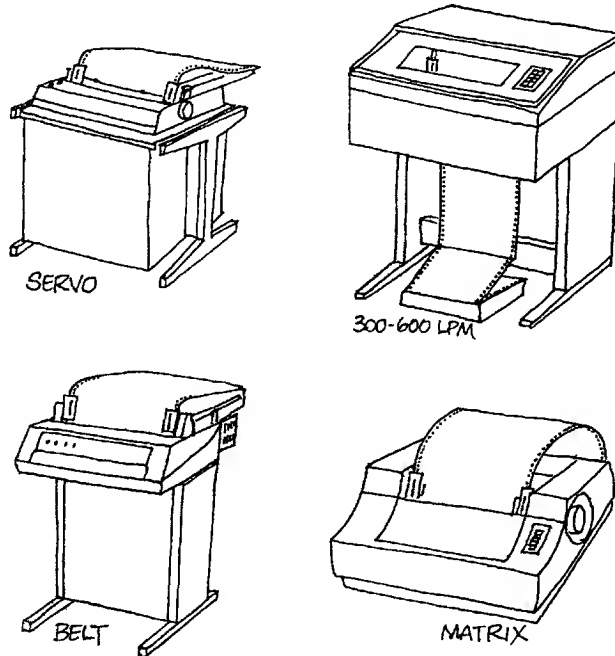
## **CHAPTER FIVE**

### **DATABUS Lesson 3 Printing the Data**



Not every Datapoint user is fortunate enough to have a printer attached to his system, but for those of you who do, this chapter will tell you how to go about getting your programming creations to use the printer.

You need to know one important thing before you design your printing program. You must know the line width of the printer you are using. The line width can be 80, 120, or 132 columns. Columns are the number of spaces across a page.



SEVERAL PRINTERS ARE AVAILABLE

So, for the moment, all you have to worry about is the number of characters across the page format of your printer unit and perhaps the availability of such goodies as a top-of-form feed, which we'll cover later.

### **The PRINT Instruction**

If you have read and digested Chapters 2 and 3, this will be familiar stuff to you. The instruction PRINT follows this general pattern of DISPLAY and KEYIN.

For purposes of discussion, we will assume you have a printer with 132 columns, and further that it is an impact printer. Impact means the printer works like a typewriter - the letters are formed by smashing metal letters coated with ink (or through a carbon ribbon) onto paper. Not that this is of serious concern to us but there must be 50 ways of making data processing printers. Some squirt liquid ink at the paper while others use a heat technique. Just be happy with what you have.

Before we write a program, consider the printer sitting next to you. Observe that there will be sprocket-holes along the sides, and the paper will be fed by moving the sprockets. Or it might have a paper

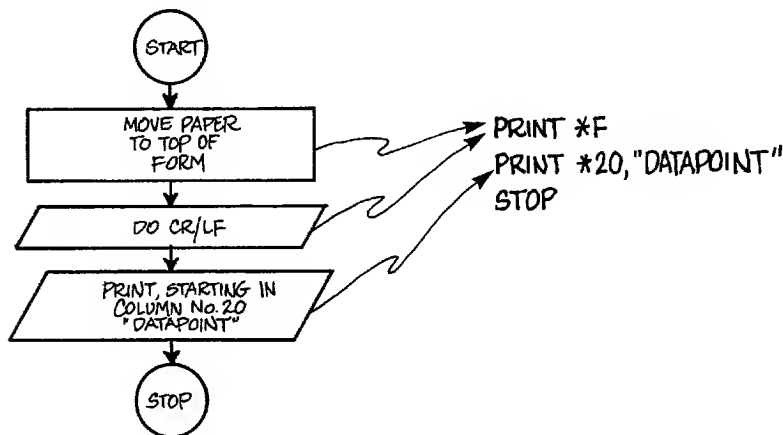
feed mechanism with a rubber roller and operate via friction as does a typewriter. Most of the expensive types with sprocket-holes contain a feature named "top-of-form". If you tell the printer to find the top of the form it will advance the paper a certain amount so that the printing mechanism will start at the upper left hand corner of the page. In most cases the paper will have perforations to delineate pages.

If you have the rubber roller friction feed type, then the top-of-form command will be meaningless to it, and you will have to position the paper manually to guarantee enough space from the tear-off.

One more thing, the last person to use the printer might not have left the print mechanism in column 1 (the left hand side of the page), so we will want to make sure we get it over there before doing anything else with the printer.

## A Printing Program

Now let's write a short program to do something with the printer. First, of course, we'll make a flow chart and then write the program.

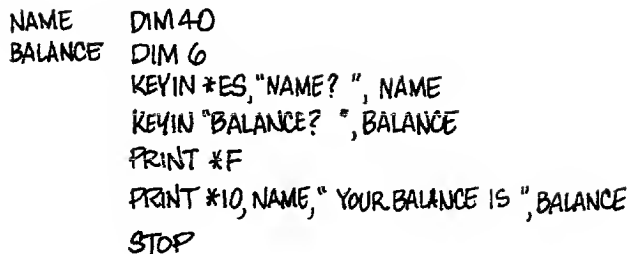


Taking the program line by line we find the `PRINT *F` instruction first. `PRINT` tells the computer to get ready to do something with the printer. `*F` is the command for top-of-form and, like the `DISPLAY` and `KEYIN` commands, it is prefaced with an asterisk. Also, like other instructions, you get an automatic carriage return and line feed unless you add the colon or semicolon to suppress them. In this case we get the `CR/LF`.

The second line again tells the computer to `PRINT` and the `*20` directs the printer to move to the right 20 spaces before printing the message. `*P` is not used with printing for positioning because the printer can only move across the page, line after line. The message is `"DATAPOINT"`.

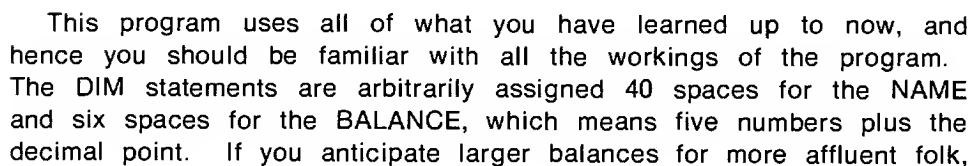
Printing is probably one of the most common data processing operations. Here is an example that could be useful to you. In this case, we want to type in a customer's name and his bank balance and have the printer type out a copy for him. While this example illustrates a limited operation, we will be adding to it later in the booklet.

## PROGRAM



NOTE: Colon feature can be used in the PRINT instruction to conserve space.  
Rules are same as in the DISPLAY instruction.

## THE DATAPOINT PROCESSOR SCREEN



the DIM for BALANCE should be increased.

KEYIN uses no new tricks, but be aware of the use of the automatic CR/LF on the printer. The next PRINT instruction spaces over 10 columns, and the name is printed beginning in column 10.

If you don't have a top-of-page feature on your printer, the printer will ignore the \*F but you will get the CR/LF anyway.

PRINT has a few other handy features such as line feed and carriage return. These are listed in the summary. See examples for proper spacing and placement of commas.

## Chapter Summary

PRINT	Allows text to be printed. The type of printer to be used with the Datapoint 2200 can be specified during Compiler and Interpreter operation.
"message"	Literal text must be bracketed in quotes
*F	Top-of-page
*L	Line Feed
*C	Carriage Return
*N	Next line (carriage return and line feed)

## Problems:

1. Write a program that prints your name at column 20 on the fourth line of the page.
2. Write a program that asks for the name and social security number and prints this on the paper.

## Solutions:

1.		PRINT	*F,*N,*N,*N
		PRINT	*20,"TOM SWIFT"
		STOP	
2.	NAME	DIM	40
	NUM	DIM	9
		KEYIN	*ES,"NAME: ",NAME
		KEYIN	"SOCIAL SECURITY NUMBER:",NUM
		PRINT	*F,NAME," ",NUM
		STOP	

## **CHAPTER SIX**

### **DATABUS Lesson 4** **Arithmetic**





While some computers spend all their time pushing names and addresses and other such data around, it's nice if they can do arithmetic too. For instance, if we had known how to add and subtract in the previous chapter, the problem involving a bank balance could have been extended to include the actual computation.

Arithmetic is especially easy in DATABUS. The four operations - add, subtract, multiply and divide - are demonstrated in the examples below:

ADD	ONE TO TOTAL	(Addition)
SUB	CHECK FROM BALANCE	(Subtraction)
MULT	DISCOUNT BY PRICE	(Multiplication)
DIV	NUMBER INTO TOTAL	(Division)

The operations are virtual English-language instructions. The labels, such as ONE and TOTAL, must contain only numbers now and are handled as special cases.

In the previous lessons, space for labels was handled with the DIM and INIT statements. DIM didn't mind if we loaded it with numbers, alphabetic characters, or punctuation.

With numeric operations, however, the only items allowed in label space are numbers and, in some cases, a minus sign.

To accomplish this numbers-only label, a new directive is used, FORM.

### **FORM Reserves Space for Numbers**

FORM allows space to be reserved for numeric characters only. The statement AGE FORM 2 allows two digits to be loaded into AGE. If you try to load any more in, a Beep will be heard and the number rejected. The statement PRICE FORM 5.2 would allow a number 5 places long to the left of the decimal point, one place for the decimal point, and two places to the right of the decimal point to be accepted. The minus sign counts as one place, also. With this in mind, PRICE FORM 5.2 holds a positive number as large as 99999.99 or a negative number of -9999.99. If the number to be used did not require a decimal point then we could have said the PRICE FORM 5. In this case, the largest value would be 99999.

FORM also allows space to be reserved and preset to some value. If the program to be written required use of the value of Pi, then we could write: PI FORM "3.14159" or FIFTY FORM "50", ONE FORM "1", and so on. You can replace these predefined numbers by loading them with other values in the program. Keep in mind the space limitations. FIFTY FORM "50" would allow a new number to replace 50 that is two numerals in length. If you anticipate replacing a preset FORM value, leave enough space, i.e., define FIFTY FORM " 50" to keep the original value the same but open up two spaces to the left of the decimal point.

### **The Arithmetic Operations**

In most arithmetic operations, one number acts upon another to form a third number, the answer. Such as:

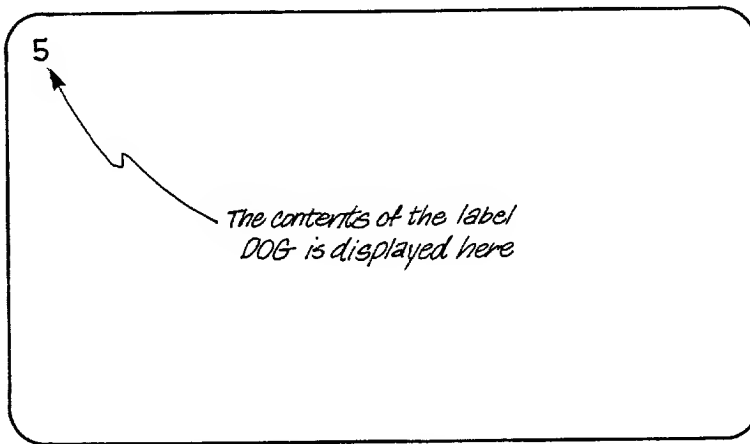
$$2 \times 3 = 6$$

$$2 \times 4 = 8$$

While there are names for the operators and operands, such as quotient and multiplicand, there's no real benefit from defining these unless you're fond of games of trivia.

DATABUS acts in the same way except that the answer ends up in the space where one of the numbers was originally. Read the example program:

```
CAT    FORM    "2"
DOG    FORM    "3"
      ADD      CAT TO DOG
      DISPLAY  *ES,DOG
      STOP
```

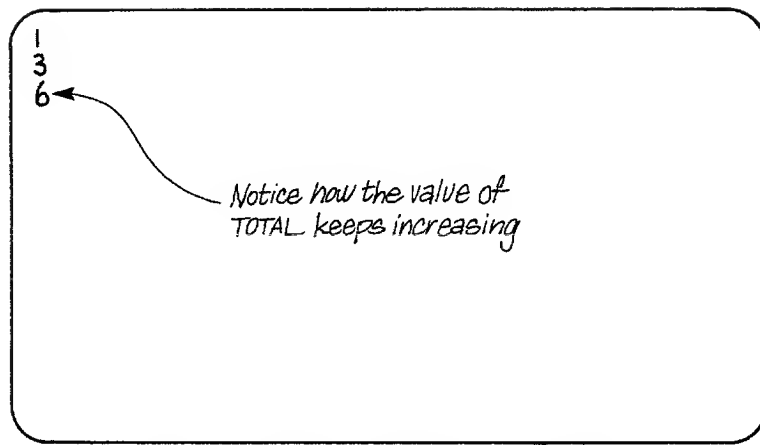


THE DATAPOINT PROCESSOR SCREEN

DOG is suddenly worth five, right? Yes, because now the answer is there. The original value of DOG, 3, is gone. In all DATABUS arithmetic operations the rightmost label contains the answer.

This trait of DATABUS can be handy in carrying totals ahead. The following example might be used in a checking account application:

```
ONE    FORM    "1"
TWO    FORM    "2"
THREE  FORM    "3"
TOTAL  FORM    2
      ADD      ONE TO TOTAL
      DISPLAY  *ES,TOTAL
      ADD      TWO TO TOTAL
      DISPLAY  TOTAL
      ADD      THREE TO TOTAL
      DISPLAY  TOTAL
      STOP
```



### THE DATAPOINT PROCESSOR SCREEN

In this example, we used TOTAL to contain the answer, and the display screen reflects the changing value of TOTAL. Notice that TOTAL was not initialized. Uninitialized values are automatically given a value of zero by the computer.

In some cases, you might need to keep both original numbers. We use a slightly different technique here involving an instruction called MOVE. In effect, the value we want to save is moved into another location (or label), so that the value we need isn't disturbed. The example below illustrates this technique:

TWO	FORM	"2"	
THREE	FORM	"3"	
FOUR	FORM	"4"	
ANSWER	FORM	2	
	MOVE	TWO TO ANSWER	= 2
	MULT	FOUR BY ANSWER	= 8
	DISPLAY	*ES,ANSWER	= 8
	MOVE	FOUR TO ANSWER	= 4
	SUB	THREE FROM ANSWER	= 1
	DISPLAY	ANSWER	= 1
	DISPLAY	TWO,THREE,FOUR	= 2 3 4
	STOP		

8  
1  
234

*The label ANSWER acted as a temporary storage place so that labels containing needed numbers weren't disturbed.*

### THE DATAPOINT PROCESSOR SCREEN

Notice that none of our preset values, TWO, THREE, or FOUR, were disturbed by the operations. ANSWER served as a temporary changeable space. We could store up to a two digit number in ANSWER, as the FORM 2 set that limit. You'll see in the display that the " 8" appears in column 2 since we said ANSWER will be two spaces wide.

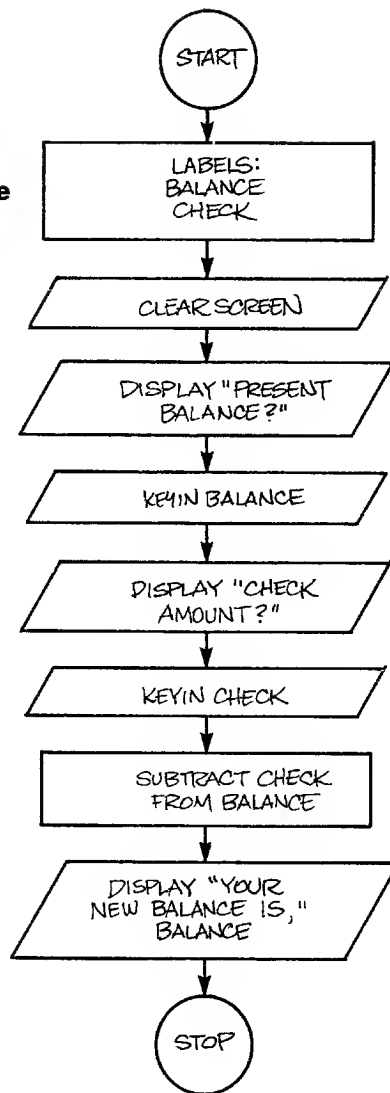
We can take the previous checkbook balance example and make it significantly more useful with arithmetic.

### PROGRAM

```
BALANCE FORM 6.2
CHECK FORM 6.2
KEYIN *ES,"PRESENT BALANCE?",BALANCE
KEYIN "AMOUNT OF CHECK?",CHECK
SUB CHECK FROM BALANCE
DISPLAY "NEW BALANCE IS $",BALANCE
STOP
```

## FLOW CHART

### Arithmetic Example



The results of this flow chart and program are shown in the following display. Keep in mind the relation of spaces to numbers as you ask DATABUS to display such things as dollar signs or question marks. You can make your programs print out as well as your local bank can!

PRESENT BALANCE? 680.00  
AMOUNT OF CHECK? 24.24  
NEW BALANCE IS \$ 655.76

*Result of checking account program.*

#### THE DATAPOINT PROCESSOR SCREEN

#### **Constants Can Also be Used**

You can also use "constant" data values in arithmetic instructions. What this means is that you can include quoted numeric values in arithmetic statements, in the following manner:

SUB        "100" FROM AMT

100 is considered a constant. Instead of initializing some variable to the value of 100, the value is just included in the instruction. This means; of course, that the value of 100 cannot be changed, it'll be 100 until you actually change the program. With label-defined numbers, as explained before, your program can change the number values living under the label name.

#### **Here are Some Precautions**

Occasionally, programmers encounter difficulties in using arithmetic instructions; two simple precautions should overcome most of these problems.

First, be certain you have allowed enough room for the result of the operation. If your result is larger than the space you have allowed, your answer will be misleading. For example, with a result of 1000 in a FORM 3 space, the stored answer will be 000; the 1 will be lost and your answer could alarm your company treasurer if you were writing an accounting program.

Secondly, when using the FORM label, only numeric characters, including decimal point and minus sign, will be accepted. This is a handy feature to minimize operator errors.

Thirdly, remember that the rightmost label value is changed to reflect the result of the computation.

## Chapter Summary

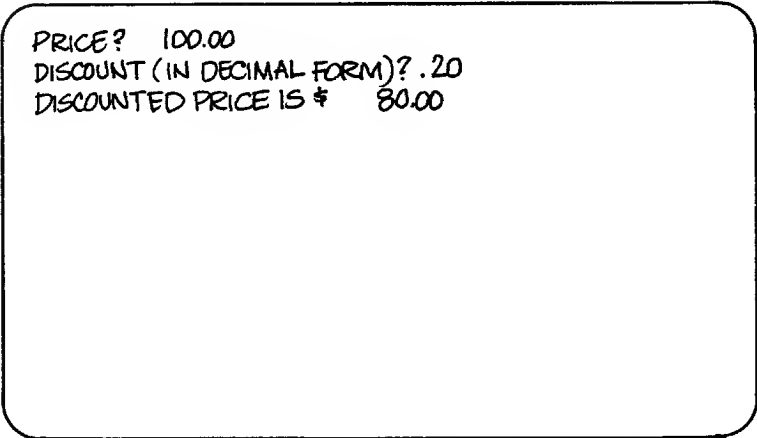
label FORM number	Allows a numeric variable of number to be stored
label FORM "number"	Allows a predetermined numeric value to be stored
MULT label BY label	Multiplication
DIV label INTO label	Division
ADD label TO label	Addition
SUB label FROM label	Subtraction
MOVE label1 TO label2	Transfer value from label1 to label2. Value transferred remains intact in first label.

### Problem:

Write a program to figure discount prices. Operator keys in price, discount rate, and sees new price on screen.

### Answer

```
PRICE      FORM      6.2
DSCOUNT    FORM      0.3
TEMP       FORM      6.2
           KEYIN      *ES,"PRICE?",PRICE
           KEYIN      "DSCOUNT (IN DECIMAL FORM)?",DSCOUNT
           MOVE      PRICE TO TEMP
           MULT      DSCOUNT BY TEMP
           SUB       TEMP FROM PRICE
           DISPLAY   "DISCOUNTED PRICE IS $", PRICE
           STOP
```



```
PRICE? 100.00
DISCOUNT (IN DECIMAL FORM)? .20
DISCOUNTED PRICE IS $ 80.00
```

THE DATAPOINT PROCESSOR SCREEN



## NOTES

## **CHAPTER SEVEN**

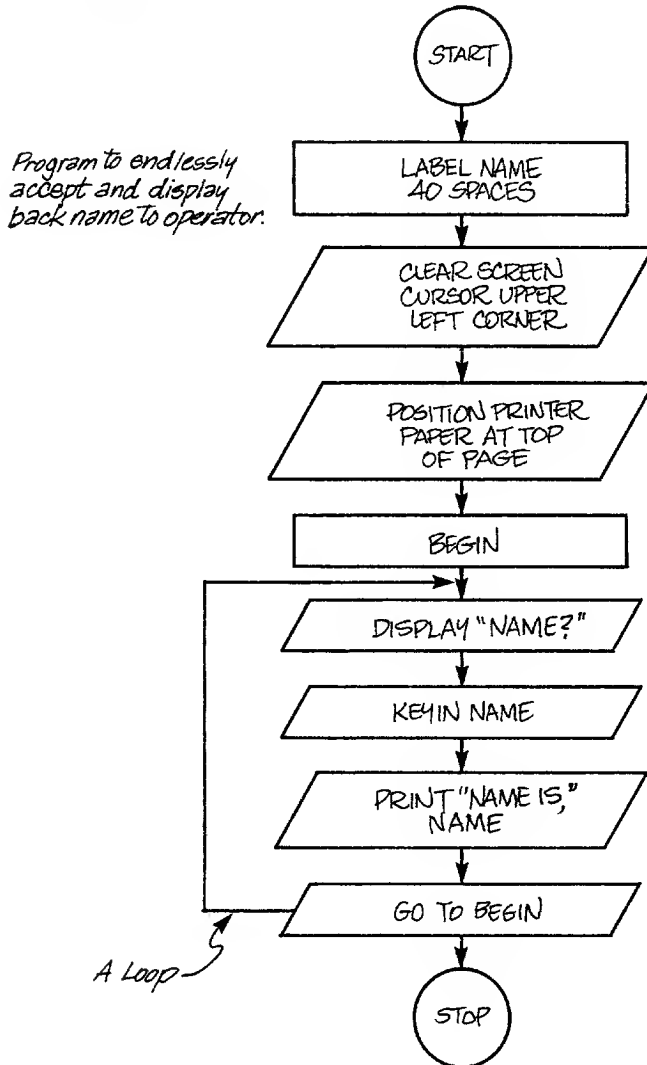
### **DATABUS Lesson 5** **Loops**



All of the programs we have written up to this point have done their job and ended. In many cases we might want to have the program repeat the task over and over. Programmers call this a loop - loops extend programming capabilities endlessly.

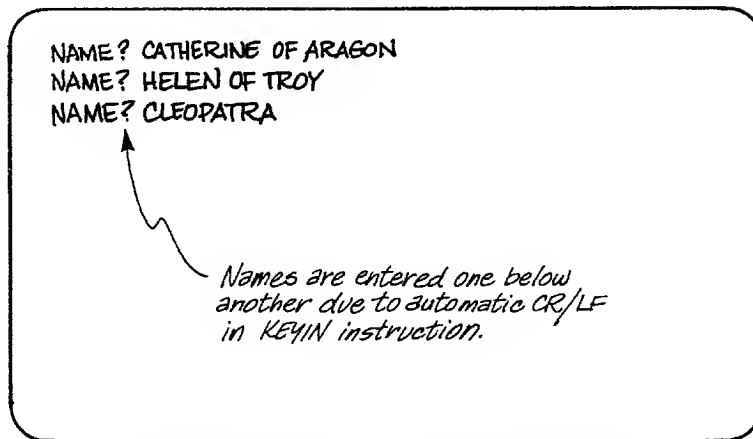
The simplest loop involves directing the program to go to a certain point and start working from there. The arrow on a flow chart can be used to indicate a jump or "GOTO" instruction. Look over the flow chart below and then read the program that follows it.

### FLOW CHART:

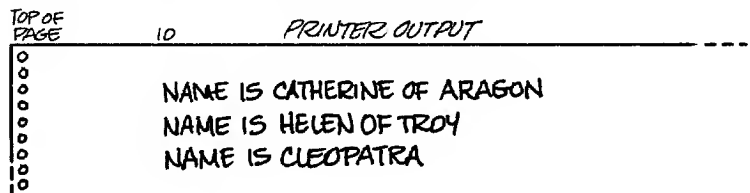


## PROGRAM

```
NAME      DIM      40
          PRINT     *F
          DISPLAY    *ES
BEGIN     KEYIN      "NAME? ",NAME
          PRINT      *10 "NAME IS ",NAME
          GOTO       BEGIN
          STOP
```



THE DATAPOINT PROCESSOR SCREEN



The GOTO instruction tells the computer to go to the place indicated by the label, in this case, BEGIN. The computer then starts following the instructions at that point. You can see that our program is endless. There is effectively no way out of this loop except, perhaps, pressing the STOP Key (which halts the computer) or by pulling the plug. Another way to halt it would be to tap the RESTART Key, which halts our program and reloads the operating system. We'll cover this in a later chapter, so don't worry about it now.

**NOTE:** If you're using a DATASHARE system you can press the INTERRUPT Key on the user terminal to halt the program.

## Conditional Loops

While endless or unconditional loops are useful in some circumstances, conditional loops are more practical. A conditional loop or conditional GOTO has limits, or goals, which must be met before the task can be done. If the conditions are not met, then the computer ignores the instruction and goes on to the next one.

One of the classic computer operations is the counting loop. In this, the programmer defines a limit, and the computer keeps on counting until the limit is reached.

To determine when the limit has been reached, a compare operation is used. The computer compares the number against the highest allowable number, the limit, and determines if they match. After each compare, the computer can tell if the number is less than, equal to, or greater than the number compared against. The computer can also compare characters.

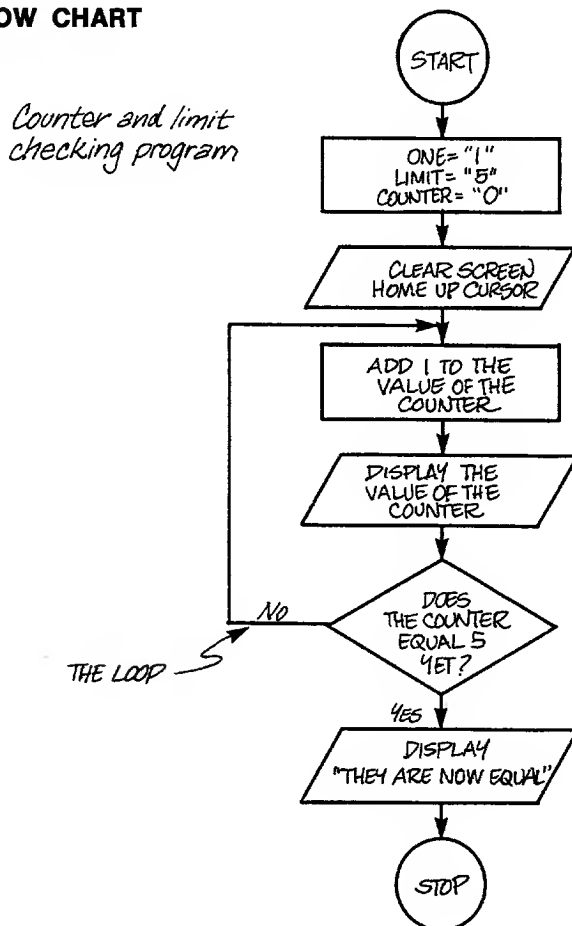
Although other, more complex compare operations are possible in DATABUS we'll confine our discussion to two types: COMPARE and MATCH.

Recall that we have used two types of data to store under labels: general mixed characters and numerics only. The DIM and INIT statements allow any type of characters to be stored under them while FORM takes only numerals. Much the same happens with the compare instructions. MATCH will compare two labels containing general text (DIM and INIT labels). COMPARE will only work with numeric labels (FORM labels).

### COMPARE is Used with Numerics

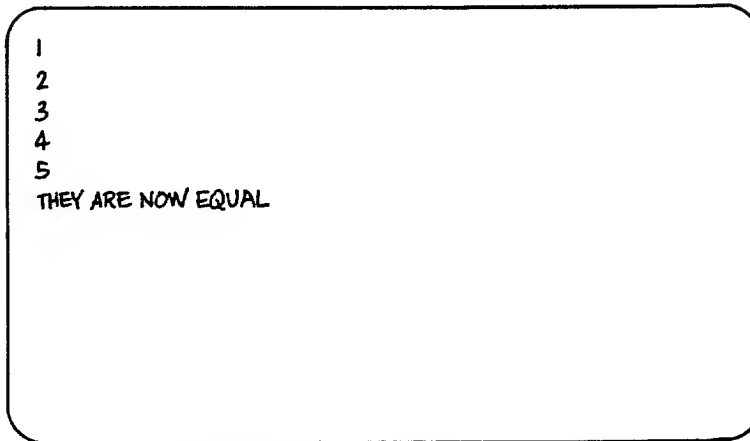
The following example illustrates the use of a COMPARE instruction with a conditional GOTO. Note that the GOTO is ignored unless the condition is met.

### FLOW CHART



## PROGRAM

```
ONE      FORM      "1"
LIMIT    FORM      "5"
COUNTER  FORM      "0"
          DISPLAY   *ES
ADDR     ADD        ONE TO COUNTER
          DISPLAY   COUNTER
          COMPARE   LIMIT TO COUNTER
          GOTO      ADDR IF NOT EQUAL
          DISPLAY   "THEY ARE NOW EQUAL"
          STOP
```



THE DATAPOINT PROCESSOR SCREEN

In this program, we started out with the label, COUNTR at zero and kept increasing it by the value of 1 until the LIMIT is equal to COUNTR. Next, if the two are not equal, the computer is asked to go back to another label, ADDR, and keep going. After five loops around ADDR, the value of five is attained, and the program ends. Remember that the COMPARE works with numerics only.

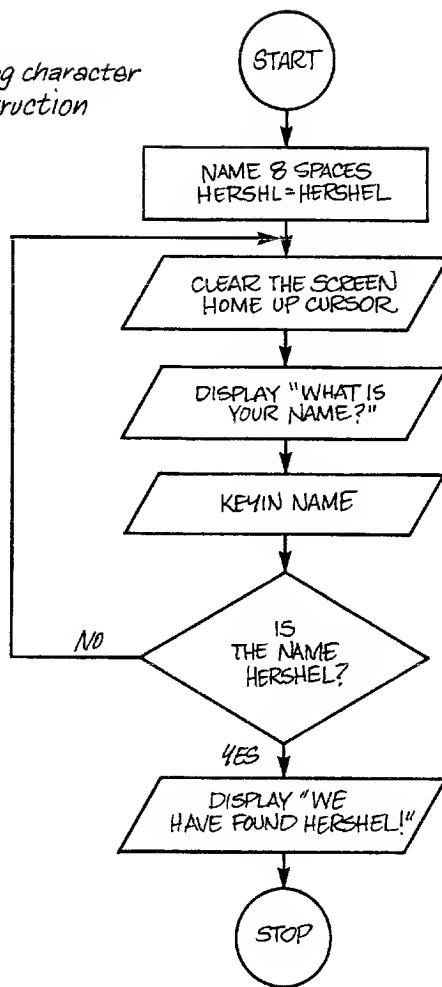
### **MATCH is Used with Characters**

While numerics are nice, you might want to use characters in the form of a name or answer. We could create an imaginary situation where people passing by are asked to type in their name and ask the computer to find someone named Hershel.

This use of general mixed characters involves the use of the MATCH instruction. This instruction can compare the contents of two character DIM or INIT labels where the COMPARE instruction worked only with numerals.

## FLOW CHART

*Program using character  
MATCH instruction*



## PROGRAM

NAME	DIM	7
HERSHL	INIT	"HERSHEL"
NUNAME	KEYIN	*ES,"WHAT IS YOUR NAME? ",NAME
	MATCH	HERSHL TO NAME
	GOTO	NUNAME IF NOT EQUAL
	GOTO	NUNAME IF LESS
	DISPLAY	"WE HAVE FOUND HERSHEL!"
	STOP	



WHAT IS YOUR NAME? HERSHEL  
WE HAVE FOUND HERSHEL!

*The question keeps on appearing  
until the correct name is keyed-in.*

### THE DATAPOINT PROCESSOR SCREEN

Note that we used the MATCH instruction. This takes the contents of the two labels and matches the first letter to the first, second to the second, and so on. But why did we test for both not equal and less? This is because there really are two things that the MATCH instruction is testing for. And while this explanation may sound a little tricky, an example should clear things up. Here's our instruction:

AB  
MATCH    A TO B

We have defined A to be "A" and B to be "AB". We find that B is equal to, but not less than A. Here's another instruction:

AB  
MATCH    B TO A

We'll use the same initial values. Here we find that A is equal to and less than B.

Obviously, MATCH only compares the same number of characters as are in the shortest character string. Here is only compared the first character for equality. But it does look at the length of both and uses the LESS signal if the second one is shorter than the first one.

### Beeping

Your Datapoint processor can make quite a racket upon command by using the BEEP instruction. It can give the user an audio clue as to what's happening.

BEEP elicits about a half second's worth of audio tone from a hidden speaker inside the processor. This tone can be used for audibly indicating the end of a job or an error.

Programming a Beep is easy. Just write BEEP.

BEEPER    BEEP  
          GOTO    BEEPER

This is a good program to use if you like to be alone. You can drive everyone screaming out of the office, as the machine will Beep

forever, although you might be kind enough to program in a way to stop the Beeping.

## Chapter Summary

GOTO label	Transfers control to the statement with the specified label
GOTO label IF condition	Transfers control to the statement with the specified label if the condition is true. Condition can be OVER, LESS, EQUAL, ZERO, EOS, NOT EQUAL NOT OVER, NOT LESS, NOT ZERO, NOT EOS
COMPARE num TO num	Compares two numbers
MATCH var TO var	Compares two character strings
BEEP	An audible beep

### Problems:

1. Write a program that examines a person's name and age. Set the program up to see if a man named GEORGE, age 28, is using the program.
2. Add the steps necessary for the program to Beep when a wrong name and age are typed in.

### Solutions:

```
1.      GEORGE  INIT      "GEORGE"
        NAME    DIM      6
        N28     FORM     "28"
        AGE     FORM     2
        LOOP1   KEYIN     *ES,"NAME? ",NAME
                MATCH    NAME TO GEORGE
                GOTO     LOOP1 IF NOT EQUAL
                KEYIN     "WHAT IS YOUR AGE, GEORGE?",AGE
                COMPARE   AGE TO N28
                GOTO     FOUND IF EQUAL
                DISPLAY   *P1:5,"YOU'RE WRONG -- SORRY!"
                GOTO     LOOP1
        FOUND   DISPLAY   *P1:5,"GEORGE,YOU'RE GETTING OLD"
                STOP
```

```

2.  GEORGE  INIT    "GEORGE"
     NAME    DIM     6
     N28     FORM    "28"
     AGE     FORM    2
     LOOP1   KEYIN    *ES,"NAME?",NAME
           MATCH    NAME TO GEORGE
           GOTO     OK IF EQUAL
           BEEP
           GOTO     LOOP1
     OK      KEYIN    "WHAT IS YOUR AGE, GEORGE?",AGE
           COMPARE  AGE TO N28
           GOTO     FOUND IF EQUAL
           DISPLAY  *P1:5,"YOU'RE WRONG -- SORRY!"
           BEEP
           GOTO     LOOP1
     FOUND   DISPLAY  *P1:5,"GEORGE,YOU'RE GETTING OLD"
           STOP

```

## **CHAPTER EIGHT**

### **DATABUS Lesson 6 Disk Operations**



Having an attached storage medium such as a diskette, cartridge disk, or mass storage disk, provides the Datapoint user with the capability to generate and maintain a group of records called a file within the system itself. In many business applications, data must be stored on disk and used by various programs.

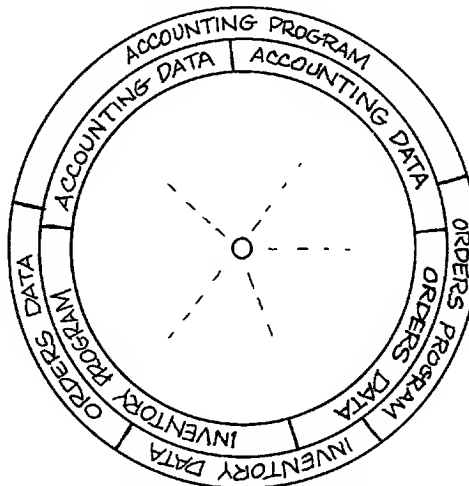
With DATABUS, we can easily record and retrieve data without regard for the usual complexities that are associated with programming a mechanical storage device. The language takes care of all of the intricate details for you.

If you want to use cassettes rather than disks for data storage, skip to Chapter 12.

Your Datapoint processor may have one of three types of disks attached to it. One outstanding feature of the DATABUS language is that, though each type of disk is structurally quite different, each is programmed in exactly the same way. What this means to you is that you can use the exact same program to write data to diskette, cartridge disk, or mass storage disk. Your programs are interchangeable between different Datapoint disk-based systems.

### **Each Disk Contains Files**

Disks are circularly shaped magnetic storage media. Data is stored in files. For example, your accounting program itself is contained in a file, and that program may read data from one or more data files. Each file has a name to uniquely identify it from the other files. You'll learn more about file names and how to create program and data files in this chapter and in Chapter 10.



**EACH DISK CONTAINS DATA AND PROGRAM FILES**

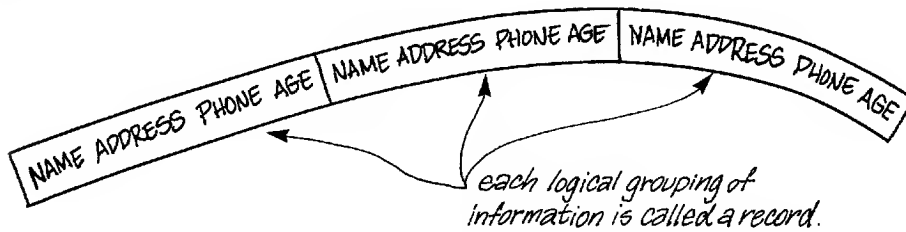
To create or use a data or program file, all you have to do is call it by name from your program. You never need to be concerned about where that program actually is on the disk — that is automatically taken care of for you by the Disk Operating System.

### **A File is a Collection of Records**

Inside each file, data is grouped into records. A record is a name

for a smaller grouping of information within one named file.

Let's look at a data file that consists of the names, addresses, telephone numbers, and ages of all of the students in a class. Each person's name, address, telephone number, and age is one record. There are as many records in that file as there are students in the class.



#### EACH FILE CONSISTS OF RECORDS

Files are divided into records to make life easier for us all. For example, to find the address, telephone number, and age of Sue Smith, all we have to do is read the beginning of each record until we find one that begins with her name.

#### It's Easy to Read From and Write to Disk

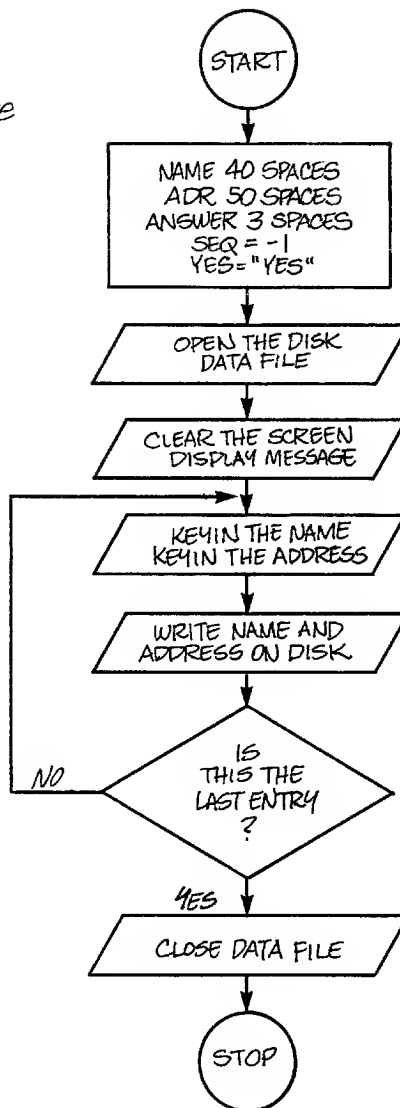
Writing data to disk and reading data from disk are actually very simple operations. Before we do an example, try to think of these operations as comparable to recording on audio tape data such information as your name, address, age, and telephone number, and all your friends' names, addresses, ages, and phone numbers. You would get the first name and details, press the record button, and say "Harry Aardvark, 149 Maple, Sod House, Idaho, 28, 684-1044". Now, release the record button and look up the next name. Later, when you're looking for an address, you would listen to the tape until the sought-for name came up and take note of the information.

In effect, we do exactly the same operation when we write to disk, except that the recording method uses digital signals rather than audio signals.

Enough of the analogies -- let's write a name and address program. First, a program to write the names on disk, then we'll discuss exactly how the program works.

## FLOW CHART

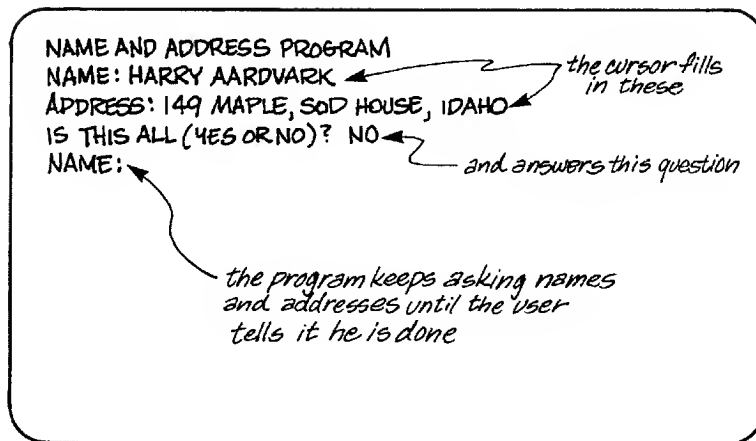
*Program to write  
data to disk*



### Program:

NAME	DIM	40
ADR	DIM	50
ANSWER	DIM	3
YES	INIT	"YES"
SEQ	FORM	"-1"
NAMEFILE	FILE	
	PREPARE	NAMEFILE,"FRIENDS"
	DISPLAY	*ES,"NAME AND ADDRESS PROGRAM"
LOOP	KEYIN	"NAME: ",NAME,*N,"ADDRESS: ",ADR
	WRITE	NAMEFILE,SEQ;NAME,ADR
	KEYIN	"IS THIS ALL(YES OR NO)?",ANSWER
	MATCH	ANSWER TO YES
	GOTO	LOOP IF NOT EQUAL
FINISH	WEOF	NAMEFILE,SEQ
	CLOSE	NAMEFILE
	STOP	





DATAPOINT PROCESSOR SCREEN

Several new things were introduced here, but don't be scared off. It's all very easy once you understand what these new instructions do. Let's look at a few sections of this program:

```
SEQ      FORM      "-1"
```

Why did we initialize this variable? Notice how it is used in the WRITE and WEOF instructions. The value of this variable tells the computer about the type of access method you are using to write your data to disk.

There are three disk writing and access methods. We're using the simplest of the three -- physical sequential. This means that the data is written to the disk exactly as your program presents it, this is, record by record. It is read back in exactly the same order as it was written, record by record.

The other two types of access methods, physical random and indexed sequential, are explained in the DATABUS User's Guides and in DATASHARE Advanced Programming Techniques.

```
NAMEFILE FILE
      PREPARE NAMEFILE,"FRIENDS"
```

In the first line, NAMEFILE is defined as the name of a file. This name is used throughout the program instructions to reference the file, yet it is not actually the real name for the file.

The real name for the data file is supplied in the PREPARE statement. PREPARE tells the computer to open a new file, name that file FRIENDS, and associate the name NAMEFILE with the file.

Why are there two names for a file -- a logical name (NAMEFILE) and a real (physical) name (FRIENDS)? DATABUS does this as a convenience to you, the programmer. Let's say that you wanted to create several data files, all with essentially the same program. Possibly you want one file for your friends in Illinois, maybe another for your friends in Kansas, and possibly a third for your friends in Iowa. Rather than changing the file name in every reference to it in the

program, all you have to do is change the real name once in the PREPARE statement. The logical name can still be the same.

This new idea of two names for a single data file may be confusing but bear with it. Remember that you only need to use the real name of the file in the PREPARE and OPEN statements. You use the logically associated name in all the other statements.

```
WRITE      NAMEFILE,SEQ;NAME,ADR
```

This tells DATABUS to write one record to the disk file. This record will contain the preset values for NAME and ADR.

```
FINISH     WEOF      NAMEFILE,SEQ
           CLOSE     NAMEFILE
```

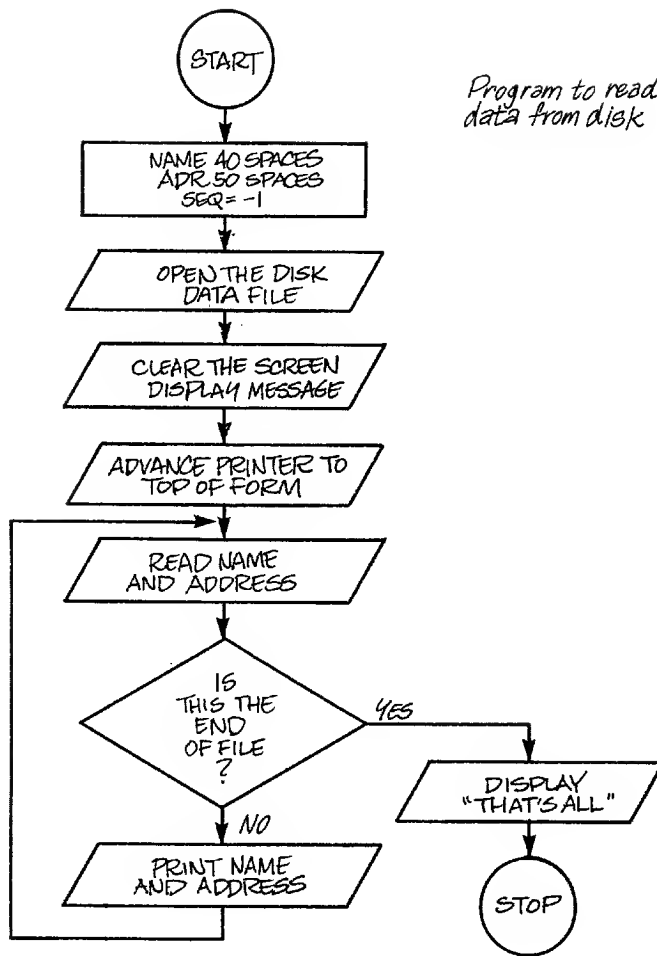
When you're through entering all of your data, you must write an End-of-File mark with the WEOF instruction. CLOSE is a signal to the computer that you're done writing all of the data for that file.

How does DATABUS know where to write the data on disk? That's the beauty of DATABUS. It takes care of all of the problems of disk space allocation for you! All you have to do is call the file by name. If there is any room left on the disk, the file can be created.

### **A Program to Read Data From Disk**

Now that you know how to write data into a data file, you need to learn how to read that data back again. Let's write a program that looks in our FRIENDS file and prints out the names and addresses that we previously wrote on the disk. Reading is a very simple operation. The data is read in exactly the same order as it is written. Look at this flow chart and program.

## FLOW CHART



### Program:

NAME	DIM	40
ADR	DIM	50
SEQ	FORM	"-1"
NAMEFILE	FILE	
	OPEN	NAMEFILE,"FRIENDS"
	DISPLAY	*ES,"NAME AND ADDRESS LISTING PROGRAM"
	PRINT	*F
LOOP	READ	NAMES,SEQ;NAME,ADR
	GOTO	DONE IF OVER
	PRINT	*N,NAME,*N,ADR
	GOTO	LOOP
DONE	DISPLAY	"THAT'S ALL"
	STOP	

See how easy it is to read data from disk? Let's go over a few new things introduced here.

```

      SEQ      FORM      "-1"
      NAMEFILE FILE
      OPEN     NAMEFILE,"FRIENDS"

```

*logically associated name* → SEQ  
*real name* → "FRIENDS"  
*logically associated name* → NAMEFILE

Again the variable SEQ is given the value of -1 to tell the computer that you're using physically sequential access. Note how SEQ is used in the READ statement.

As with the example that wrote data to disk, NAMEFILE is used as the logical file name and FRIENDS is the actual physical file name. While NAMEFILE is the name we use to refer to our data file throughout our program, FRIENDS is the actual name of the data file. If we wanted to use this same program to read similar data about another group of friends from a file named FRIENDS1, all we'd have to do is change the OPEN statement to OPEN NAMEFILE,"FRIENDS1".

OPEN looks just like the PREPARE statement in the writing example. We use OPEN to open an existing data file and PREPARE to open and create a new data file.

```

      LOOP      READ      NAMES,SEQ;NAME,ADR
                GOTO      DONE IF OVER

```

*logically associated name* → NAMES  
*access method* → SEQ  
*label* → DONE  
*end of file* → OVER

Each name and address is read from the disk data file. How can we tell when we've read them all? When we try to read beyond the end of the data in the file, the OVER condition is set. That's why the second statement is included. We check to see if the OVER condition has been set by the read, and we tell the program to go to the statement with the DONE label if it is.

In WRITE and READ operations, you get back what you put in, in the same order. Always check for the End-of-File mark as soon as the disk is read so you know when you're done reading the data.

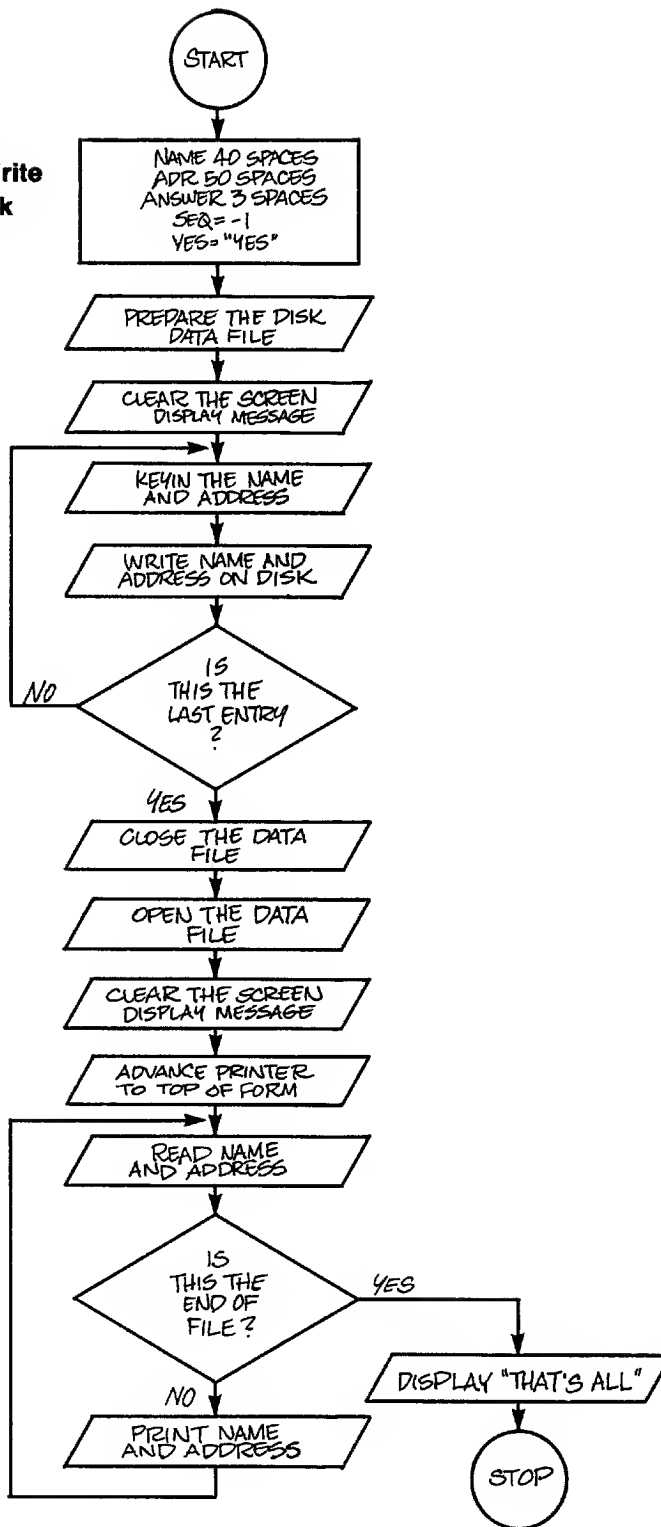
### A Program that Writes and Reads

You can easily combine the two previous examples to write data to disk, then read and print it. You can use this program to proofread the names and addresses you put in the file.

Look at the flowchart -- it's just a combination of the two previous flowcharts:

## FLOW CHART

To Read and Write  
Data on a Disk



Now look at the program:

```
NAME      DIM      40
ADR       DIM      50
ANSWER    DIM      3
YES       INIT     "YES"
SEQ       FORM     "-1"
NAMEFILE  FILE
          PREPARE  NAMEFILE,"FRIENDS"
          DISPLAY  *ES,"NAME AND ADDRESS PROGRAM"
LOOP1     KEYIN    "NAME: ",NAME,*N,"ADDRESS: "ADR
          WRITE    NAMEFILE,SEQ;NAME,ADR
          KEYIN    "'IS THIS ALL (YES OR NO)?",ANSWER
          MATCH    ANSWER TO YES
          GOTO     LOOP1 IF NOT EQUAL
FINISH    WEOF     NAMEFILE,SEQ
          CLOSE    NAMEFILE
          OPEN     NAMEFILE,"FRIENDS"
          PRINT    *F
LOOP2     READ     NAMEFILE,SEQ;NAME,ADR
          GOTO     DONE IF OVER
          PRINT    *N,NAME,*N,ADR
          GOTO     LOOP2
DONE      DISPLAY  "THAT'S ALL"
          STOP
```

Wasn't it easy to combine the reading and writing programs to make this program that proofreads the file? Notice how we've already covered every element of this program in our previous explanations.

### **Adding Data to the File**

Since we all like to think of ourselves as interesting people, we should make some provisions to add new friends' names and addresses to our data file. Let's think out this process before we flow chart it.

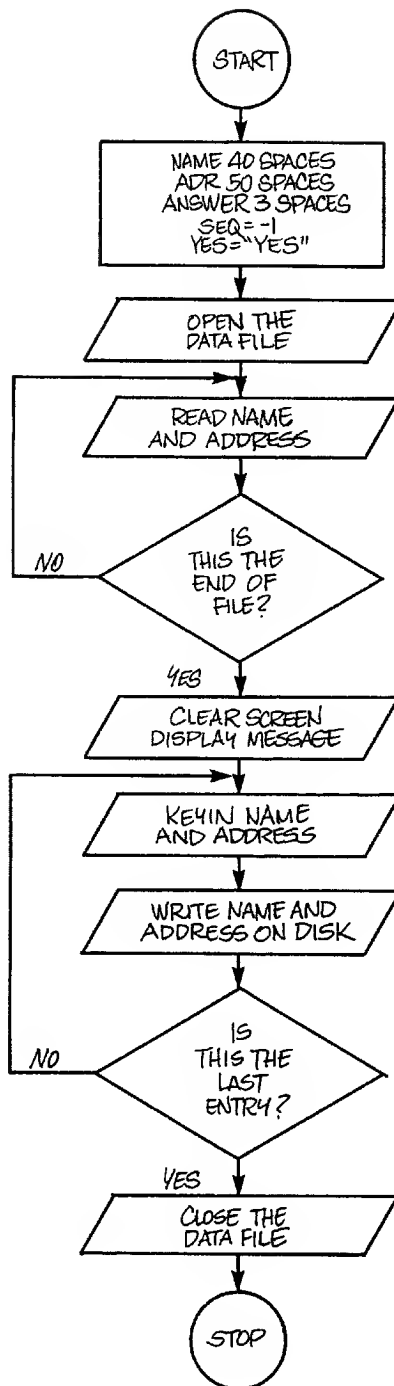
Our data file is full of names and addresses of other friends. We still want these, so first of all we should read to the end of the file so we don't overwrite anyone's information.

Once we've read to the end of the file, we need to start writing the names and addresses of our new friends. Once we've added all the new names and addresses, we need to close the file again.

Here's a flow chart of our program that adds data to the file.

## FLOW CHART

### Adding Data to the File



Here's the program:

NAME	DIM	40
ADR	DIM	50
ANSWER	DIM	3
YES	INIT	"YES"
SEQ	FORM	"-1"
NAMEFILE	FILE	
	OPEN	NAMEFILE,"FRIENDS"
	DISPLAY	*ES,"FILE ADDITION PROGRAM"
LOOP	READ	NAMEFILE,SEQ;NAME,ADR
	GOTO	LOOP IF NOT OVER
DONE	KEYIN	"NAME:",NAME,*N,"ADDRESS: ",ADR
	WRITE	NAMEFILE,SEQ;NAME,ADR
	KEYIN	"IS THIS ALL(YES OR NO)?",ANSWER
	MATCH	ANSWER TO YES
	GOTO	DONE IF NOT EQUAL
FINISH	WEOF	NAMEFILE,SEQ
	CLOSE	NAMEFILE
	STOP	

## Chapter Summary

label	FILE
-------	------

Label is a logical file name.

PREPARE	logical,physical
---------	------------------

Creates a new file with the specified physical file name and associates the logical file name with it.

OPEN	logical,physical
------	------------------

Opens an existing file with the specified physical file name and associates the logical file name with it.

CLOSE	logical
-------	---------

Close the file.

WEOF	logical,access
------	----------------

Write an end-of-file mark on the file, using the specified access method.

WRITE	logical,access;variables
-------	--------------------------

Write one record of the file according to the access method.



Read one record of the file according to the access method.

**Problem:**

1. Write a program that reads the name and address file and prints out only selected names and addresses.

**Solution:**

```

NAME1    DIM      40
NAME      DIM      40
ADR       DIM      50
ANSWER    DIM      3
YES       INIT     "YES"
SEQ       FORM     "-1"
NAMEFILE  FILE
          PRINT     *F
LOOP      OPEN     NAMEFILE,"FRIENDS"
          KEYIN     *ES,"NAME: ",NAME1
LOOP1     READ     NAMEFILE,SEQ;NAME
          GOTO     BAD IF OVER
          MATCH    NAME1 TO NAME
          GOTO     LOOP1 IF NOT EQUAL
PRINT     PRINT     *N,NAME,ADR
          KEYIN     "IS THIS ALL?",ANSWER
          MATCH    ANSWER TO YES
          GOTO     DONE IF EQUAL
          GOTO     LOOP
          DISPLAY  "NOT IN FILE -- TRY AGAIN"
DONE      DISPLAY  "THANKS"
          STOP

```

## **CHAPTER NINE**

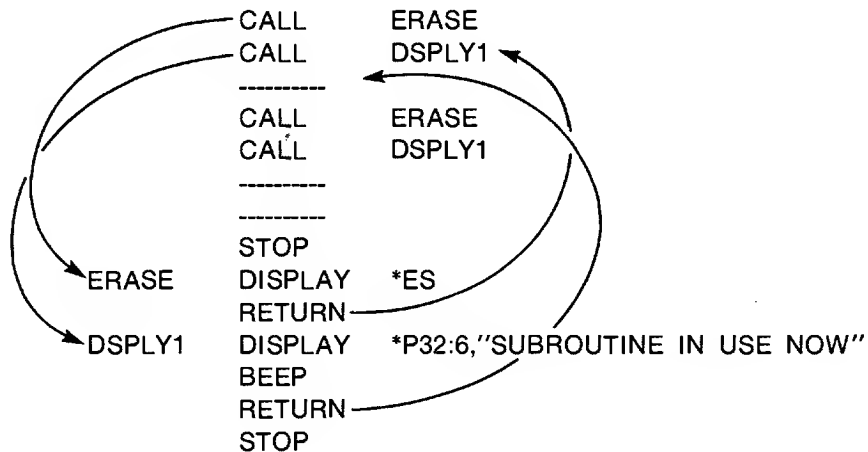
### **DATABUS Lesson 7 Using Subroutines**



Often a user finds that a particular sequence, or grouping, of instructions crops up quite frequently in the programs he writes. For instance, in a data entry application a user may frequently want to erase the screen, display a form, Beep and do related chores. To have to write out this identical sequence of instructions each time is burdensome and time consuming. Fortunately there exists a handy technique for the elimination of this drudgery. This involves the labeling of these standard sequences that reappear in program writing with distinctive names and making them available upon request to the program writer.

These standard sequences of instructions are known as subroutines. A user can call for a subroutine when required, have the repetitive task at hand accomplished, then get back to the mainstream of the program. To do this we make use of the CALL and RETURN instructions.

To demonstrate this useful feature of DATABUS, let's assume that you're writing a program that is quite lengthy and which requires you to erase and place various messages upon the screen. The example has no actual application but it will serve to give you the hang of the subroutine approach. In the following example, note that the CALL instruction jumps to the instruction containing the label and keeps working on that line of instructions until it encounters the RETURN instruction. At that point, the computer jumps back to the instruction immediately following the CALL instruction. The dotted lines indicate instructions that are not pertinent to the example.



The arrows show the "leaping around" the computer goes through in finding and executing the subroutines. You can CALL other subroutines even though you're in one already. CALLS can be "nested" eight deep. That is, you can say CALL eight times before saying RETURN. If you nest the subroutines more than eight times, the computer will lose track of what's going on and weird things will happen, so avoid that condition.

Naturally, the advantage of this subroutine feature lies in being able to write shorter programs and occupy less space in the computer's memory. (Memory economy assumes that you use the subroutine more

than once or the space saving benefit won't hold true.)

The purpose of a subroutine is to avoid repetition. Instead of coding a group of the same instructions over and over in your program, you group those instructions into a subroutine and call that subroutine whenever you need it. Your subroutines can be as elaborate as you want, containing any DATABUS instruction.

### **Linking Up with Other DATABUS Programs**

In some applications, DATABUS programs can be so large that they occupy the entire memory space of the Datapoint processor. Since Datapoint disk-based systems use a virtual memory technique, the program can be actually much longer than the physical memory, but some programs might occupy all this space. In these cases, it is desirable to have one program fetch another as soon as it is finished. This eliminates the need for a user to wait around and see if a part of a job has finished, so that he can proceed with the next part.

To do this operation, called "chaining", we make use of the CHAIN instruction. To use chaining, just code the CHAIN instruction at the appropriate part of your program. For example,

```
CHAIN    "PGM2"
```

chains to PGM2. PGM2 is the name of the other DATABUS program you want to execute. This other program must be on the disk in computer readable form -- but don't worry too much about that now, you'll learn how to set up your programs in Chapter 10. This could also be coded as:

```
NEXT      INIT      "PGM2"  
          .  
          .  
          CHAIN      NEXT
```

in cases where the name of the program to be chained to is a variable. Chaining instructs the computer to go out and find and execute the program name specified.

### **Common Data Areas**

You can even carry forward information in labels as you chain from one program to another. Let's suppose that your first program asked for today's date via a KEYIN instruction and you wanted to use this date in all other programs you were going to run. Note that asterisks are used to define what has been carried over from the previous program.

Program No. 1			Program No. 2		
MONTH	DIM	2	MONTH	* DIM	* 2
DAY	DIM	2	DAY	* DIM	2
YEAR	DIM	2	YEAR	* DIM	2
PGM2	INIT	"PGM2"		-----	
	KEYIN	MONTH,DAY,YEAR		-----	
	-----			-----	
	-----			-----	
	-----			-----	
	CHAIN	PGM2		-----	
	STOP			STOP	

This data that you carry over from one program to another is called common data, for obviously the values of that data are common to both programs.

A word of caution about using common data areas between programs -- notice that the common variables are coded in exactly the same order in both programs. You must keep the common data area in both programs in exactly the same order or the computer will confuse the values of the variables.

Here are the rules for carrying labeled values from one program to another:

1. The program being chained to must be a compiled DATABUS program.
2. Be sure the common variables are in the beginning of each of the programs and in the same order.
3. Use the asterisk in the labels to denote that the computer is to carry this value along to the next program.
4. FILE statements cannot be passed in common.

## Chapter Summary

	CALL	label	Transfers operation to instruction indicated by label.
	RETURN	Label	Transfers operation to instruction immediately following last CALL instruction.
	CHAIN	Label	Locates, loads, and runs named DATABUS program.
Label	DIM	*n	Asterisk allows label data to be carried from one program to another.
Label	INIT	*n	
Label	FORM	*n	

## NOTES

## **CHAPTER TEN**

### **Running Your DATABUS Program**





If you've gotten this far, you probably have a program scratched out on the back of an envelope that you would like to try, or perhaps you might first try one of the examples in this booklet.

In any case, arm yourself with the following items:

1. A Datapoint Processor (1100, 2200, or 5500).
2. Some kind of Datapoint printer (not absolutely essential, but very handy).
3. The DATABUS compiler. We highly recommend that you use the latest release of DBCMP. As of this writing the diskette DBCMP is model code number 20264 and the cassette DBCMP is model code number 20263 (cartridge and mass storage disk users should get the cassette version -- we'll show you how to load and run it on disk).
4. The DATABUS interpreter. Again, get the latest version. DB11ASYS, model code number 20217, is the cassette tape that can be loaded on a cartridge disk system. DB11BSYS, model code number 20218 is the cassette tape that can be loaded on a mass storage disk system under DOS.B. DB11SYS, model code number 20219 is the diskette containing the interpreter for diskette systems.

The sequence of events used to take the program from a concept to an actual running computer program is fairly involved, yet simple. Read and follow the directions carefully -- success will be yours.

But before fame and glory are yours, we have to get the program in the machine and running. The following table lists the sequence of events.

1. Get the DBCMP compiler and the DB11 interpreter on disk or diskette.
2. Type in the program. (Editor)
3. Convert your program into computer code. (Compiler)
4. Run it using the interpreter. (Good Luck)
5. Doesn't run the way you would like? Fix it.(Editor)  
Back to Step No. 2.

## **Step 1**

### **The DBCMP Program**

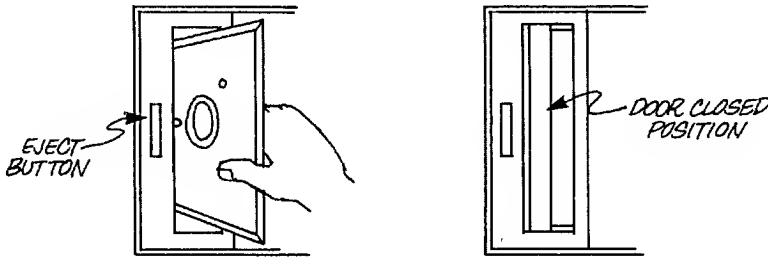
#### **If You've Got a Diskette System....**

The DBCMP diskette contains not only the DATABUS compiler, but also the DOS.C operating system. You'll probably want some extra diskettes to store all of the programs that you will be writing and for backup copies of those programs.

Turn your diskette system on. If you are unfamiliar with Datapoint equipment, see the Guide to Operating Datapoint Equipment. Insert the DBCMP diskette in drive 0 (the left-most drive).

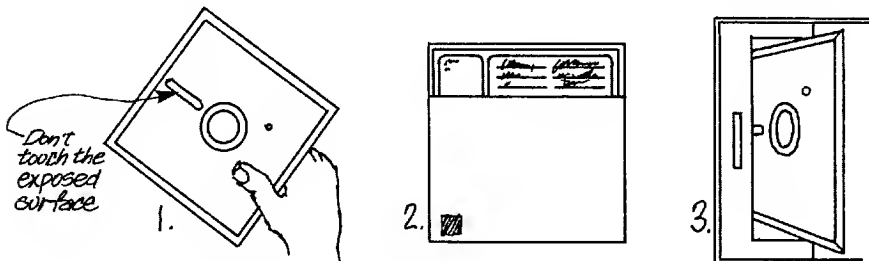
To insert the diskette, slide the door open on Drive 0 and slide the

DBCMP diskette in until you hear a faint click and the diskette is securely locked in. Be sure the diskette faces the correct way (see the drawing). Then slide the black door over the slot opening. The diskette is now loaded and ready to use.



### INSERT THE DISKETTE IN THE CORRECT POSITION

Before we go any further, here's a few words of caution about diskette handling. Do not lay the diskette down anywhere if it's not in its envelope, as it picks up dust and lint which could cause errors.



### THE ONLY THREE PLACES FOR A DISKETTE

There are only three places for a diskette -- in your hand, in the drive or in the envelope.

Now, to get your system in working order, press the RESTART Key, which is located on the far right side of the keyboard. This begins the "DOS" program, which enables you to run other programs. After pressing the RESTART Key the screen will look like this:



DOS is now waiting for input. Keep the DBCMP diskette in drive 0 and put the DB11SYS diskette in drive 1 (the next diskette drive). You'll need the DB11SYS diskette for Step 4. Go to Step 2 to see how to create your DATABUS program.

### If You've Got a Cartridge or Mass Storage Disk System...

Turn your Datapoint system on, put your disk in place, put a DOS boot tape in the rear cassette deck, and press RESTART (or RESTART and RUN on a 5500). If you're unfamiliar with Datapoint equipment, see the Guide for Operating Datapoint Equipment. You'll know that the system is ready when the operating system displays "READY" on the screen.

To see if you've already got the DBCMP compiler on your disk, type "CAT DBCMP↵" (↵ means to press the ENTER Key). If the system responds with "DBCMP/CMD" and "READY", you have DBCMP on your disk and you do not have to put it on your disk.

READY  
CAT DBCMP  
DBCMP/0V0 (200) DBCMP/0V1 (203) DBCMP/0V2 (204) DBCMP/CMD (176)  
READY

*this asks if DBCMP is on the disk*

*this response shows that it is on the disk*

DATAPOINT PROCESSOR SCREEN

If, however, DBCMP is not already on your disk, you must put a copy of the DBCMP cassette in the front cassette deck and type "MIN ;AO↵". This will put a copy of the compiler on your disk.

READY  
CAT DBCMP  
READY  
MIN ;AO

*this asks if DBCMP is on the disk*

*it isn't*

*this puts the contents of the front cassette deck (DBCMP) on the disk*

DATAPOINT PROCESSOR SCREEN

Type "CAT DB11↵" to see if the DATABUS interpreter is on your disk. If the processor responds by displaying a list of files and "READY", the interpreter is on your disk. If not, put the DB11 cassette in the front cassette deck, type "MIN ;AO↵". This will put a copy of

the interpreter on your disk.

Now you're ready to type in your program.

## Step 2

### Type in the Program

First of all, you must decide on a name for your program. Pick something significant so that you can remember it. The name must start with an alphabetic character and can be a total of eight characters long. For our examples, we'll call our program NAMES.

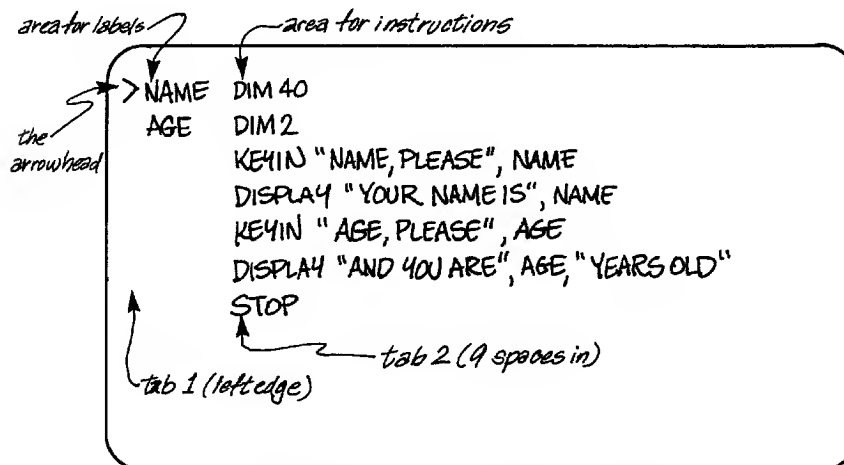
We need to use the Editor program to help us create our program. By the way, the Editor program has nothing to do with green eye shades and cub reporters. The Editor, in the computer business, is a program allowing text entry (your program) and the later modification of the text, in case you blow it.

To call the Editor program, type "EDIT NAMES;D)". We need to supply the name of the program, NAMES, and that we want to write a DATABUS program, D.

The ";D" tells us that we want to use the DATABUS preset tabbing facility. To make the job easier, the Editor program has preset tab stops for labels and instructions. Tab 1 is located at the left edge where the labels are typed. Tab 2 is located about 9 spaces from the left edge, where instructions begin. If you tap the space bar while you're in the first tab area (columns 1-8), you'll automatically be skipped over to the ninth column, where you can begin typing your instructions (see the illustration below).

The screen will clear and there will be an arrowhead pointing at one of the blank lines. To enter new lines of your program, just type them in, pressing ENTER after each line. While in the line, you can use the BACKSPACE Key to correct errors and the CANCEL Key to start again from the last tab stop.

The drawing below shows how our display screen looked when we typed in our program:



DATAPOINT PROCESSOR SCREEN

What if you make a mistake? Notice the small arrowhead in the left-most column. Press the KEYBOARD and DISPLAY Keys one at a time and watch the arrowhead (line pointer) move up and down. By fiddling with these two keys, you can point out any line with an error.

Now, with the arrowhead pointing to the line containing the error, you can type in the Editor commands to get at the error. All Editor commands are prefaced with a colon. Unless you begin each command with a colon, the Editor will assume it's just another line of the program. Remember that the arrowhead must be pointing (each time) to the line which is to be changed and you must press the ENTER Key after typing in the Editor command. Here are three of the most versatile Editor commands:

:DEL	(Delete) Blots out the entire indicated line and lets you type it in again or leave it out entirely.
:INS	(Insert) Opens up a space between two lines so that another line may be squeezed in.
:MOD	(Modify) This command lets you change individual characters or a group of characters. Suppose you accidentally typed DSPLAY and didn't notice it. Point the arrowhead at the line and type ':MOD DSPLAY<DISPLAY'. The Less Than (<) Symbol is placed between the old and new characters. This line will now contain the proper word.

:MOD      DSPLAY<DISPLAY  
                  ↑      ↑      ↑  
                  *(old)* *(replaces)* *(new)*

There are several other Editor commands that you can use (see Appendix B for a complete list). The commands may be shortened to one letter, if desired, i.e., :MOD can be :M.

All of this will be somewhat hazy until you have some experience at hammering away at the keyboard. One last step -- when you've got everything so it looks good, stop and contemplate this opus magnus. Then type the last and most important Editor command:

:END	(End of Program) Indicates to the Editor that you are done with the program and that you want to write a copy of your work on disk or diskette. If you forget it, the program will be lost.
------	---

After :END has been typed in, the Disk Operating System message and "READY" will reappear on the screen. Now you're ready for Step 3.

### Step 3

#### Compile Your Program

Translating your DATABUS instructions into machine instructions is a very simple process. All you need to do is type "DBCMP name)" where name is the name of your program. In our case, we'll type "DBCMP NAMES)". The compiling operation adds a computer readable version of the text on the disk or diskette. In effect, there will be two files on the disk or diskette after compiler operation. One file will contain the text of the program you typed in; the other file will contain the computer readable code.

If all goes well and you haven't made any language mistakes (the compiler, of course, can not check for logical errors), your Datapoint processor display screen should look very similar to the following illustration:

```
READY
DBCMP NAMES
DOS. DATABUS COMPILER -1.1- 29 AUG 75
PASS 1
  20.
PASS 2
  20.
DOS.A DATAPoint CORPORATION'S DISK OPERATING SYSTEM 1.1
READY
```

THE DATAPoint PROCESSOR SCREEN

#### Error Messages

However, if you've made a mistake an error message will be displayed.

```
DBCMP NAMES
DOS. DATABUS COMPILER -1.1- 29 AUG 75
PASS 1
  20.
PASS 2
  6. 100120 STATE DIN
ERRORS WERE 1
DOS.A DATAPoint CORPORATION'S DISK OPERATING SYSTEM 1.1
READY
```

THE DATAPoint PROCESSOR SCREEN

The error messages are really very easy to understand. In this case, 6 tells us that the error occurred in line 6 of the program. It tells us the type of error (an undefined instruction). 00120 tells us the octal position of that instruction in the program (you can usually ignore this). And STATE DIN is the instruction in error.

In this case, it is fairly easy to figure out where we went wrong. We spelled DIM wrong. Now we need to go back to Step 2, to correct the program using the Editor.

Sometimes, however, the error won't be so obvious. You may need a printout of your program to find out exactly what your problem is. To get a listing with the compilation, type "DBCMP NAMES;L)". The L specifies that the entire program should be listed on the printer. The compiler will ask you for a heading for your listing. The heading can be up to 70 characters long and say whatever you like. Often it's a good idea to include the date and time for future reference.

Here's a copy of our program with the error:

1.	00000	DONE	INIT	"YES"
2.	00010	ACNT	DIM	5
3.	00020	NAME	DIM	20
4.	00047	ADR	DIM	20
5.	00076	CITY	DIM	15
6.	100120	STATE	DIN	2
7.	00121	ZIP	DIM	5
8.	00131	AFILE	FILE	
9.	01001		OPEN	AFILE,"SAMPLE1"
18.	01147	FIN	DISPLAY	*P1:3,"THANKS"
19.	01162		STOP	
20.	01163		STOP	

ERRORS WERE I

There are three basic types of errors. An I error indicates an undefined instruction. U indicates an undefined variable or label. D means duplicate label. And E indicates a general syntax error. In the case of E errors, a number is given on the line with an asterisk.

You can get more information about error messages in the DATABUS Reference Manual.

#### Step 4

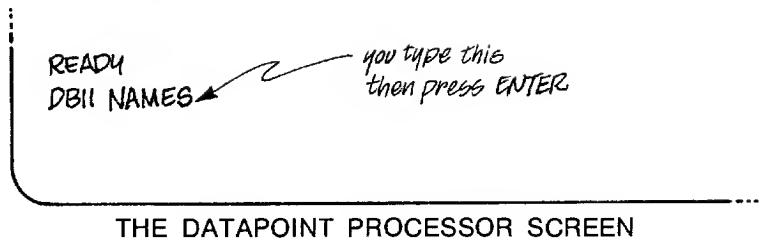
##### Run Your Program

Now that your program seems to work (at least all of the syntax problems have been resolved), you need to execute it. Then you'll get to see how your program, which is now compiled into computer-readable form, really works. If you're using a Diskette system, be sure that the DB11SYS diskette is in drive 1.

To execute the program, use the DATABUS interpreter. Type "DB11 name)" where name is the name of your program. In our case, we



type "DB11 NAMES)".



Soon the display screen will clear and you'll see any visual displays from your program. If, by chance, you typed in the name of a nonexistent program, the screen will clear, you'll hear a Beep, and then the Disk Operating System message and "READY" will reappear.

### Step 5

#### Fix Your Program

Did your program do what you wanted it to do? Are there any minor enhancements you would like to make? Now that you've seen how your program works, do you want to change it?

If you want to change it, go back to Step 2. Edit your program, compile it again, and then execute it. Hopefully you'll like it this way.

Congratulations -- it worked! Wasn't it easy? We're sorry that we had to take a lot of the mystique of the world of computers away from you, but you really don't have to tell your friends how easy it is to use a Datapoint computer.

#### General Hints

1. Try an example program in the book to get the hang of all steps.
2. Make sure the printer is turned on if you're going to use it.
3. If you have an unexplainable problem, give your Datapoint Systems Engineer or Account Manager a call and they'll help you with your problem.

**CHAPTER ELEVEN**  
**For DATASHARE Users**



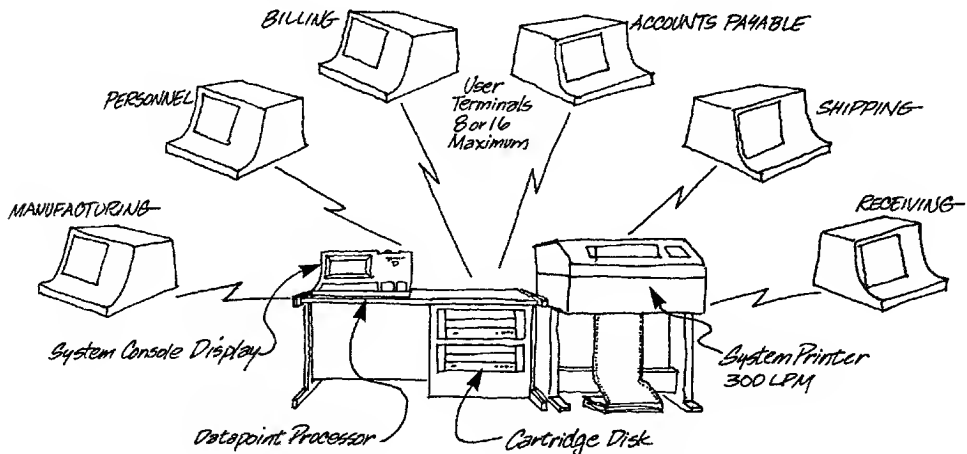
## What is DATASHARE?

DATASHARE is Datapoint's multi-user, multi-task, business data entry and processing system. DATASHARE allows many simultaneous users to share the full capabilities of a Datapoint 2200 or 5500 processor. Many different programs can be executed simultaneously. One user may be updating inventory, another running payroll, and still another doing order entry.

Programs, written in the powerful DATABUS language, can be shared among some or all users. They can be restricted to certain users through the use of security passwords. For example, payroll files would most likely be restricted to a very limited number of users while inventory files would probably be available to a wider range of employees.

DATASHARE allows local or remote users. Some DATASHARE users may be in the same room as the processor while other users are located across the country.

A DATASHARE system consists of a Datapoint 2200 or 5500 processor, at least one cartridge or mass storage disk, a printer, and up to 8 or 16 user terminals. You can't use a Diskette 1100 as a DATASHARE processor.



A TYPICAL SIX-USER DATASHARE SYSTEM

## The DATABUS Compiler

You use DBCMP to compile the DATABUS programs that you will be running under DATASHARE. This is the same compiler that you use for DATABUS programs that you don't want to run under DATASHARE. What this means is that you do not have to recompile a DATABUS program to be able to run it under DATASHARE. See Chapter 10 for instructions on how to use DBCMP.

## The DATASHARE Interpreter

The DATASHARE Interpreter is a program that runs on the central processor and manages the execution of the compiled DATABUS programs on each user's port. The DATASHARE Interpreter that you use for your system depends on the equipment that you have. Use the following chart as a guide to the Interpreter for your system.

Processor	Disk	Terminals	Interpreter	User Terminals
2200	Cartridge 1K Controller	3360	DS3A3360	8
2200	Cartridge 1K Controller	3600	DS3A3600	8
2200	Cartridge 4K Controller	3360	DS3B3360	8
2200	Cartridge 4K Controller	3600	DS3B3600	8
2200	Mass Storage	3360	DS3B3360	8
2200	Mass Storage	3600	DS3B3600	8
5500	Any	Any	DS35500 DS3RFILE PSDS	16

There are some significant things to note about this chart. First of all, you cannot mix 3360 and 3600 terminals on a 2200 DATASHARE system. Also, because of compatibility problems, we do not recommend the use of non-Datapoint terminals.

On a 5500 DATASHARE system, however, it is possible to mix terminals; 3360's and 3600's may be on the same system.

There are three DATASHARE interpreters you can use on a 5500 system. DS35500 is the standard DATASHARE interpreter and is similar in operation to all of the interpreters used for 2200 systems, discussed in this book.

The two special 5500 DATASHARE interpreters, DS3RFILE and PSDS, are not covered here but are covered in their respective user's guides and in *Advanced Techniques in DATASHARE -- A Simplified User's Guide*. DS3RFILE is used when Diskette 1100's are used as user terminals and diskette files, as well as DATASHARE files, are accessed. PSDS, the Partition Supervisor DATASHARE, allows for concurrent DOS and DATASHARE operations.

### **Put the Programs on Your Disk**

In this section we'll only cover the software end of setting up a DATASHARE system. We'll assume that all of the cables have been connected and that your equipment is ready to get to work. If you are not acquainted with Datapoint equipment, see the *Guide to Operating Datapoint Equipment*.

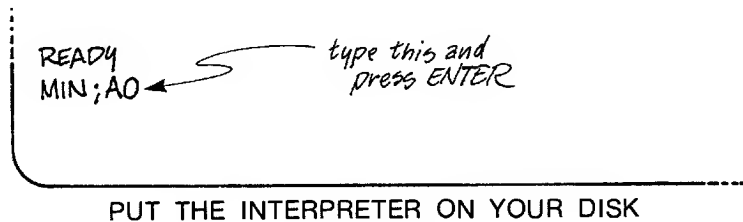
The previous chapter discussed how to compile (translate your program into machine readable code) and run DATABUS programs. You should compile all of the DATABUS programs you have ready now, before you set up your DATASHARE system. Remember to test these programs out before you let your operators incorporate them into their daily routine.

Let's assume that you've got your programs written for the various

departments or offices that will be using your DATASHARE system. Your inventory control program is flawless and your accounting programs are works of art.

Now you should be sure that the proper interpreter is on your disk pack. Choose the proper interpreter for your system and check to see if it is there by typing "CAT DS3A3600" and press the ENTER Key, or substitute the interpreter name you chose instead of DS3A3600.

If you don't have the interpreter on your disk, get a copy of it on cassette, put the cassette in the front cassette deck, type "MIN ;AO" and press ENTER, as shown below. This will put a copy of the interpreter program on your disk.



READY  
MIN;AO

*type this and  
press ENTER*

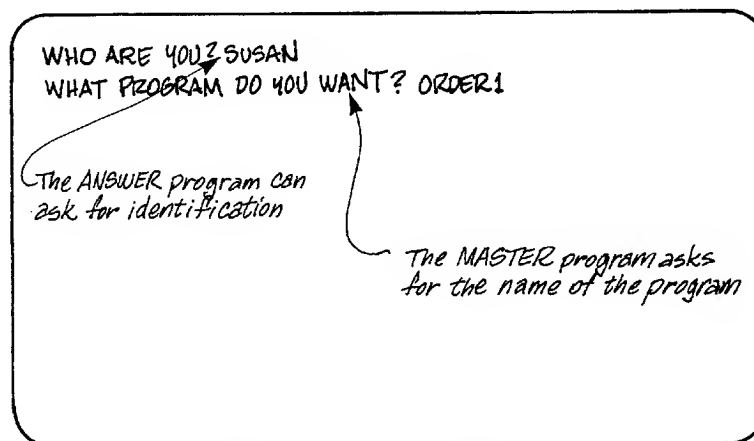
PUT THE INTERPRETER ON YOUR DISK

### Set Up ANSWER and MASTER Programs

But before you can start operating your DATASHARE system, you've got one more step to take. Before a DATASHARE system can be used, there must be an ANSWER and MASTER program for every port. Both of these programs are written in the DATABUS language. From the lessons in this book you have already learned everything you'll need to write these programs.

The ANSWER program is the program that each user encounters when he first gets on to the system. The ANSWER program gives you the option of requiring that the user properly identify himself before using the system.

The MASTER program is executed after ANSWER and after each DATABUS program is executed. The MASTER program usually requests the name of the program that the user wants to execute.



WHO ARE YOU? SUSAN  
WHAT PROGRAM DO YOU WANT? ORDER1

*The ANSWER program can  
ask for identification*

*The MASTER program asks  
for the name of the program*

THE DATAPOINT PROCESSOR SCREEN  
ANSWER AND MASTER PROGRAMS

Some DATASHARE users develop fancy and quite ingenious ANSWER and MASTER programs, and you may end up being quite creative yourself. But for starters, it isn't necessary to be complicated at all. As a matter of fact, it's really easy to set up simple ANSWER and MASTER programs.

### The ANSWER Program

The user must satisfy the requirements of the ANSWER program before he can gain access to the DATASHARE system. See Advanced Techniques in DATASHARE -- A Simplified User's Guide for details on security procedures. We'll just give you an ANSWER program that we created. Though it's simple, it should give you a good idea of what you may want to write.

```

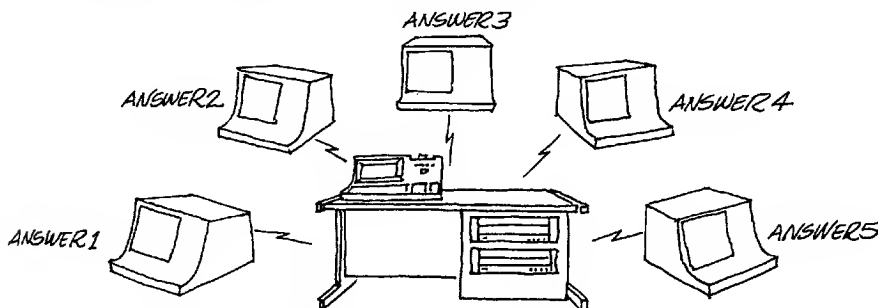
TERMN  FORM  13
ID      DIM   10
IDCODE  INIT   "DATAPOINT"
        DISPLAY *ES,"DATASHARE PORT",TERMN
        CONSOLE "ANSWER",TERMN
LOOP    KEYIN  "ID:",ID
        MATCH  ID TO IDCODE
        GOTO   GOOD IF EQUAL
        DISPLAY "****INVALID ID****"
        GOTO   LOOP
GOOD    STOP
        STOP

```

There is one new instruction here -- CONSOLE. It works just the same as DISPLAY, but displays the message at the system console. The system console is the 2200 or 5500 processor display screen.

What this program does is to display the terminal number at the user's terminal and at the console, and then ask for a code word. If the identification matches, the program is satisfied and the MASTER program is automatically executed. If the identification doesn't match, the user has to try to type it in again.

See how this program displays which user terminal it is? Each port should have its own ANSWER program. The ANSWER program for terminal one must be named ANSWER1, for terminal two is ANSWER2, etc. If an ANSWER program does not exist for a specific terminal, a program named ANSWER, if it exists, will be executed.



EACH TERMINAL SHOULD HAVE ITS OWN ANSWER PROGRAM

However, if you aren't that worried about system security, you can avoid writing separate programs for each terminal by writing one universal ANSWER program, naming it ANSWER.

The MASTER program is executed after each DATABUS program, including ANSWER, under DATASHARE. The MASTER program asks the name of the program that the user wants to execute, then executes that program.

What should your MASTER program look like? Look at this program:

Once you understand the new instructions we've introduced here, it's easy.

We'll skip the TRAP instruction momentarily, and move along to CHAIN. CHAIN transfers control from your program to another DATABUS program. In this case, this is how we get out of the MASTER program and begin executing a program of the user's choice. CHAIN was explained in Chapter 9.

Let's look closely at the TRAP instruction:

91



CFAIL is the event that we are trapping. CFAIL is an abbreviation for CHAIN FAIL -- that is, the program name specified in a later CHAIN instruction is not on the disk.

NONAME is the label that is on the statement that control is transferred to if a CFAIL occurs. You can use any label name that you want. In our program, NONAME is the label attached to the error message, "\*\*\*NO SUCH PROGRAM\*\*\*".

There are other events that we can TRAP. Because most of these are rarely encountered by the beginning programmer, we won't explain them here. They are explained, however, in the DATABUS User's Guide.

Our MASTER program simply asks for the name of a program and then executes it. You may want to be more selective, and only let the user execute one of a specified list of programs. You can find more detail about the MASTER program in Advanced Techniques in DATASHARE -- A Simplified User's Guide.

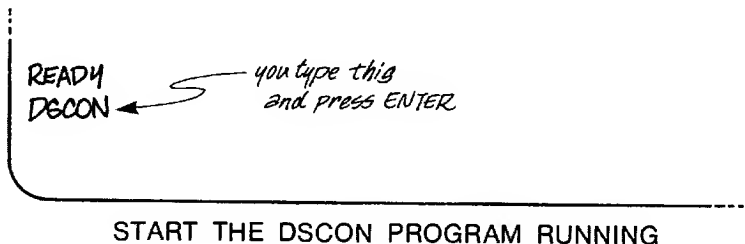
### The DSCON Program

Let's review the steps you should have taken by now to get your DATASHARE system ready:

1. You should have compiled all of the DATABUS programs your operators will be using.
2. You should have tested these programs out to be sure that they work.
3. You should have picked out the proper interpreter and made sure it is on your disk.
4. You should have written an ANSWER and MASTER program for each terminal.

Your next step is to tell the DATASHARE interpreter exactly how you want your DATASHARE system set up. To do this, you have to run the DSCON program, which already is on your disk (it came with the interpreter). DSCON stands for DATASHARE configurator.

DSCON asks you a series of questions about the configuration of your system. To run DSCON, type "DSCON" and press ENTER.



You will then be asked a series of questions about your DATASHARE system. If your system already has been configured, the configuration will be displayed on the screen for you, and you will be asked if you want to change it. If you answer "YES", you will be asked these questions, too.

Here is the way the screen looked when we answered the DSCON questions:

```
ENABLE ROLLOUT? YES
SERVO PRINTER? YES
PORT 1 ON CONSOLE? YES
BYPASS MULTI-PORT ADAPTOR? NO
NUMBER OF PORTS? 3
EQUAL DATA AREAS FOR ALL PORTS? NO
3823 BYTES LEFT. PORT 1 DATA AREA SIZE: 2000
1823 BYTES LEFT. PORT 2 DATA AREA SIZE: 1000
823 BYTES LEFT. PORT 3 DATA AREA SIZE: ALL THAT IS LEFT

READY
```

### THE DATAPOINT PROCESSOR SCREEN

### DSCON QUESTIONS

Let's look at each question individually.

#### ENABLE ROLLOUT?

ROLLOUT is a program that allows a DATASHARE user to temporarily stop DATASHARE execution at all the terminals and execute DOS commands. Once the DOS commands have been executed, DATASHARE can be restored to its previous status. ROLLOUT is fully explained in Advanced Techniques in DATASHARE -- A Simplified User's Guide and in the DATABUS User's Guide.

You will probably want to answer "YES" to this question simply because you rarely will need to use this facility, but when you do, you won't have to run DSCON again to change the configuration record.

#### SERVO PRINTER?

This question asks if you've got a Datapoint Servo Printer attached to your processor. If you have any other type of printer, such as a 300 line-per-minute printer, or no printer at all, answer "NO".

#### PORT 1 CONSOLE?

Do you want to use the Datapoint processor screen as a monitor of all DATASHARE activities? If not, you can use the processor as terminal 1 of your DATASHARE system by answering "YES" to this question.

#### BYPASS MULTI-PORT ADAPTOR?

You should answer "NO" unless you plan to have only one terminal -- the system console. You will need to use the multi-port adaptor to connect to other terminals.

#### NUMBER OF PORTS?

Here you type in the total number of terminals your system will be

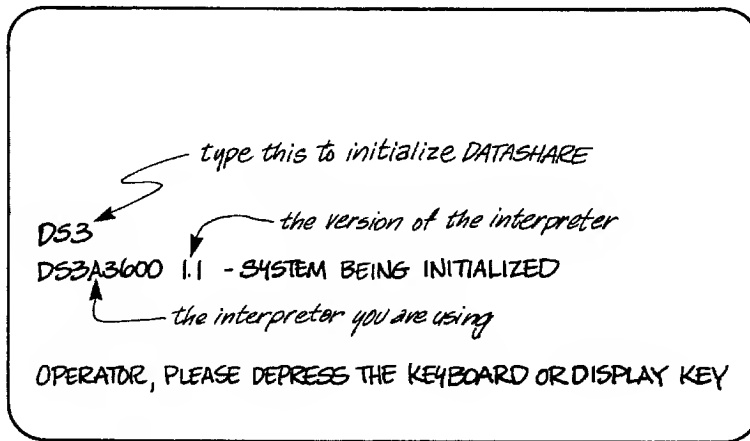
using. This number includes the system console, if you have that configured as a terminal. This number can be from 1 to 8 if you're using a 2200 processor, or from 1 to 16 if you're using a 5500 DATASHARE program.

#### EQUAL DATA AREAS FOR ALL PORTS?

If you answer "YES", each user will be given the same amount of space in the processor for data manipulation and storage of declared variables (the variables you declared in DIM, INIT, and FORM statements). If you know that certain terminals will be running programs requiring extremely large data areas, you can individually assign the amount of data area available to each terminal, as we did in our example. However, because there is so much data area available, you probably will just want to divide it equally among all terminals.

#### DATASHARE System Operation

Now you're ready to start using your DATASHARE system. This section explains how to initiate DATASHARE using the DS3A3600, DS3A3360, DS3B3600, DS3B3360, and DS35500 interpreters. See Advanced Techniques in DATASHARE -- A Simplified User's Guide for details on the DS3RFILE and PSDS interpreters.

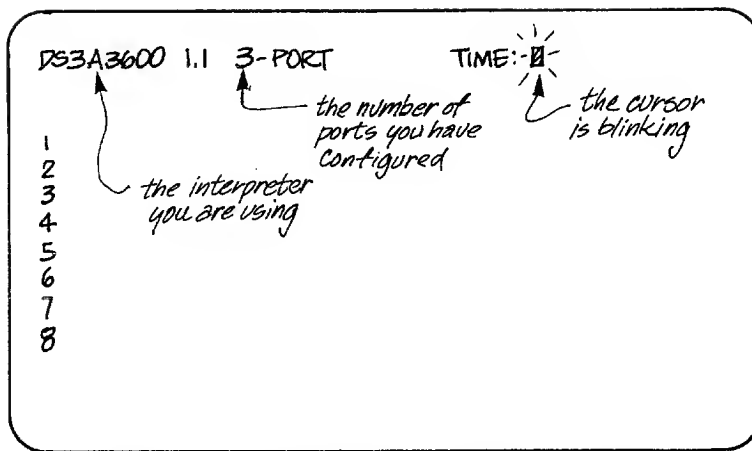


DATAPoint PROCESSOR DISPLAY SCREEN

This is what the screen of our processor looked like when we initialized our DATASHARE system. Notice that all we had to do was type "DS3" and press ENTER to start running the system. (If you're using DS35500, type "DS35500" instead of "DS3").

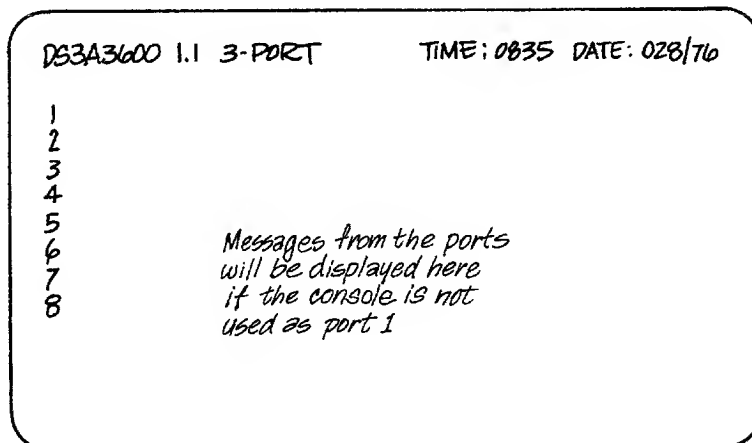
Press the KEYBOARD or DISPLAY Key to show DATASHARE that an operator is present. If one of these keys are not depressed within 30 seconds, the processor will make a series of one second warbles to try to attract your attention. After 30 seconds of Beeping, the racket will stop and it will be assumed that DATASHARE is being operated in unattended mode. You want to operate in attended mode most of the time so you can interactively relate to DATASHARE.

If there is an operator present, however, there's one more step. After pressing the KEYBOARD or DISPLAY Key, your processor screen should look like this:



DATAPOINT PROCESSOR DISPLAY SCREEN

Enter the time according to a 24 hour clock and press ENTER. Enter the DATE as the three digit Julian date, a slash, and the last two digits of the year. Then press ENTER. Your screen should look similar to this:



DATAPOINT PROCESSOR DISPLAY SCREEN

If terminal 1 is the console, the processor screen will clear and the ANSWER program for terminal 1 will be executed.

Now all of the terminals can connect to the main processor, using direct wire connections or telephone connections. Each user will have to satisfy the ANSWER and MASTER program requirements before he can start executing his programs.

### Taking Down the System

You should use ROLLOUT to temporarily stop DATASHARE execution and return to DOS. You use ROLLOUT to do such things as edit programs and index data files. ROLLOUT is explained in Advanced Techniques in DATASHARE -- A Simplified User's Guide. ROLLOUT allows you to return to DATASHARE operation without reinitializing the system.

Before you completely bring your DATASHARE system to a halt, make sure that each user is done with his programs. If you interrupt a program in the middle of execution, you won't do any irreparable damage, but you will make life difficult. For example, if you've only let an operator type in a third of the data for one program, that operator may have to type it all in again or write a program to let him pick up from where he stopped.

To stop DATASHARE execution and return to the Disk Operating System, simply press RESTART at the processor console.

## **CHAPTER TWELVE**

### **DATABUS 2 - For Cassettes**



For those of you without disks attached to your Datapoint processor, Datapoint has another version of DATABUS that will work with cassettes.

In this chapter, DATABUS 2 (Version 5) is described. If you are using an older version of DATABUS 2, we highly recommend that you order the newest compiler and interpreter available.

The first step in learning how to use cassette DATABUS is to read all of the lessons in this book except lesson 6. Since lesson 6 covers disk file structures, and you aren't using disks, it is unnecessary to read that lesson.

This appendix covers three major subjects:

1. Language differences between disk DATABUS (DATABUS 11) and DATABUS 2.
2. Data storage and retrieval on cassette tape.
3. Running a DATABUS 2 program.

Once you've learned the basics described in this appendix, you're probably going to want to learn more about DATABUS 2. See the DATABUS 2 User's Guide, model code number 50169, for more details.

## Part 1 - Language Differences

This section covers the differences between DATABUS 2 Version 5 and disk DATABUS. The differences covered relate to the subjects already discussed in this book. Check the DATABUS 2 Version 5 User's Guide for complete language specifications.

### Label Differences

DATABUS 2 labels can only be six characters long, while labels can be up to 8 characters long in disk DATABUS. Other than that, the rules are the same.

#### Good Labels

COMP  
A123  
CALBAK


#### Labels That Won't Work

COMPUTER (too long)  
123A (starts with a number)  
!!NOTE (illegal first character)

### KEYIN and DISPLAY Differences

There are two important differences in the display screen positioning controls under DATABUS 2:

1. There is no ~~\*ES~~ parameter in DATABUS 2. Instead, to position the cursor to the upper left-hand corner of the screen and erase the screen, use the following DISPLAY instruction:

DISPLAY \*P1:1,~~\*ES~~ 

Remember that \*EF just erases the screen from the current cursor position -- in this case, the upper left-hand corner. You must position the cursor via the \*P command.

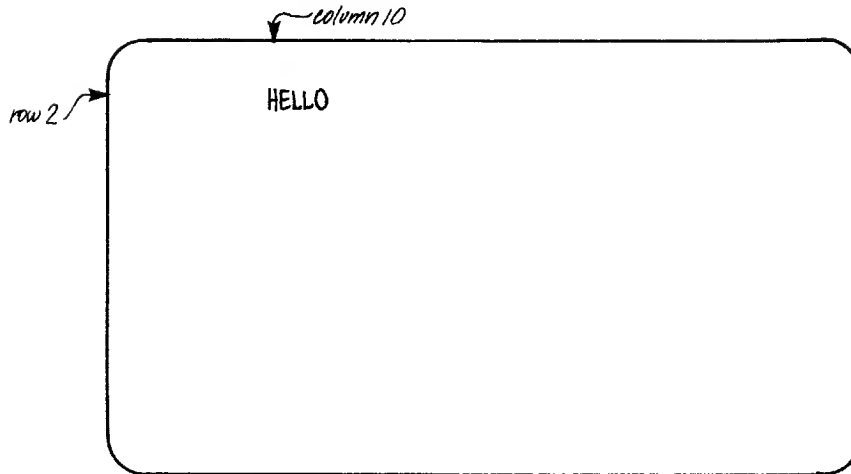
2. In addition to \*P positioning, \*V and \*H can be used to specify



the vertical and horizontal cursor positions. The following two DISPLAY statements have the same results:

```
DISPLAY *V2,*H10,"HELLO"
```

```
DISPLAY *P10:2,"HELLO"
```



DATAPOINT PROCESSOR SCREEN

These are some of the valid cursor positioning controls under DATABUS 2:

Control	Function
*Hn	Causes cursor to be positioned to column n ( $1 < n < 80$ ).
*Vn	Causes cursor to be positioned to row n ( $1 < n < 12$ ).
*Px:y	Causes cursor to be positioned to column x and row y.
*EL	Erases the rest of the line from the current cursor position.
*EF	Erases the rest of the screen from the current cursor position.
*R	Rolls the screen up one line, losing the top line and sets the bottom line to blanks.

### Constant Data Values

Remember how, in the disk DATABUS lessons, you could use this statement:

```
ADD      "100" TO AMOUNT
```

to add the 100 to the value of AMOUNT? 100 is considered a

constant value and is never changed by the program.

Well, in DATABUS 2, you cannot use constant data in this way. Instead, you need to give the value of 100 a labeled name, and use that label in the arithmetic statement instead of the constant value. This requires two statements, a FORM statement to initialize the value:

```
HUND      FORM      "100"
```

and the arithmetic statement:

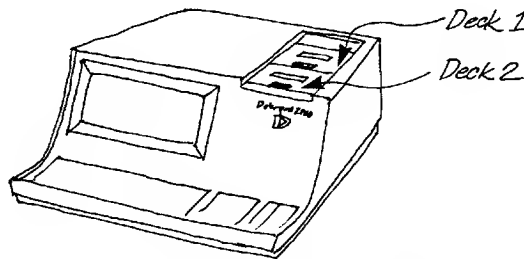
```
ADD      HUND TO AMOUNT
```

### **Data Storage Statements**

For obvious reasons, the disk data storage statements (discussed in Lesson 6) are not included in DATABUS 2. Instead, a whole new set of commands are used to read and write data to and from cassettes. These are discussed in the next section.

### **Part 2 - Data Storage on Cassette Tape**

Each Datapoint processor has two cassette tape decks. The deck towards the back of the processor is called deck 1; the deck near the front of the processor is called deck 2.



THE CASSETTE DECKS ARE NUMBERED

The use of cassette tapes involves only several logical steps and adherence to three rules.

### **A Few Guidelines**

These guidelines are important to remember for effective data file reading and writing.

First, the data cassette tape should be rewound before it is used in order to assure a common beginning place. Otherwise, your program could begin reading or writing the cassette at a place that will be troublesome to find again.

Secondly, after writing all your data, be sure to tell the tape that this is the end of the data. A WEOF (Write-End-of-File) mark should be written.

Thirdly, controls should be set up while reading data so that the End-of-File mark can be detected to notify the program that there's no more data.

## Writing Data to Cassette Tape

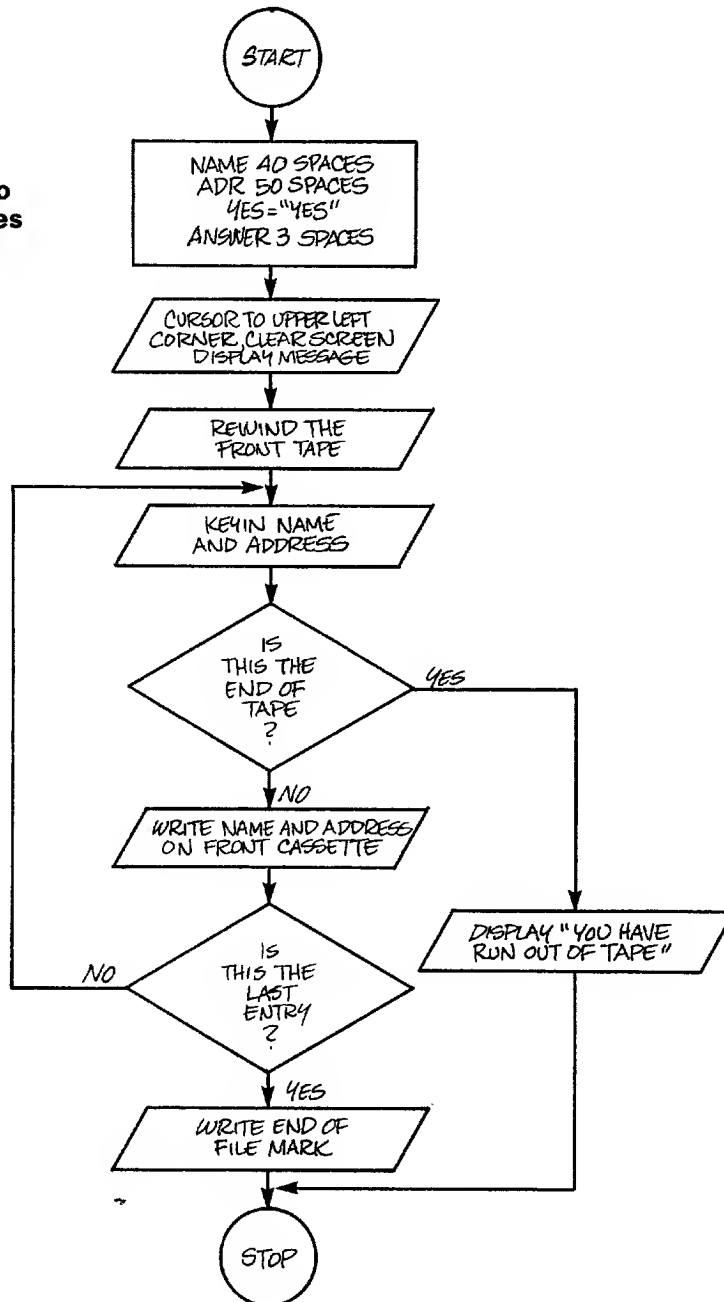
You use the WRITE instruction to write data to cassette. We'll go through an actual program example to show how easy writing data to cassettes is.

Let's look at the program we used to write data to disk, but this time we'll write the data to cassette tape. Remember that the program creates a file with the names and addresses of friends.

Here's the flow chart:

### FLOW CHART

#### Program to Write Data to Cassette Tapes



And here's the program:

NAME	DIM	40	
ADR	DIM	50	
ANSWER	DIM	3	
YES	INIT	"YES"	
	DISPLAY	*P1:1,*EF,"NAME AND ADDRESS PROGRAM"	
	PREPARE	2	
	TRAP	ENDTAP IF EOT2	←
WRITER	KEYIN	"NAME: ",NAME	
	KEYIN	"ADDRESS: ",ADR	
	WRITE	2,NAME,ADR	
	KEYIN	"IS THIS THE LAST ENTRY? ":	
		"ANSWER YES OR NO",ANSWER	
	MATCH	ANSWER TO YES	
	GOTO	FINISH IF EQUAL	
	GOTO	WRITER	
ENDTAP	DISPLAY	"YOU'RE OUT OF TAPE"	
	BKSP	2	
FINISH	WEOF	2	
	BEEP		
	STOP		

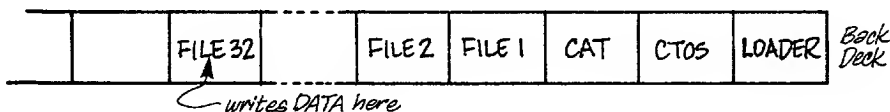
Let's look at some of these new instructions:

PREPARE 2

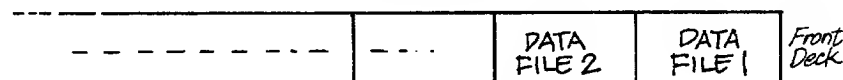
This tells DATABUS to rewind cassette deck 2, the front cassette deck. If there is already data on the front cassette, this operation will render it inaccessible and ready the cassette for a new data file.

If your instruction would have been PREPARE 1, a data file would be opened on the same cassette that contains the interpreter and program (see Part 3 for instructions on how to run your program).

When you write data on the rear cassette deck, it is put in file 32, after the interpreter, your programs, and any other files. See the drawing below.



However, when you write to the front cassette deck, the data files begin right at the beginning of the cassette.



TRAP        ENDTAP IF EOT2

TRAP was explained under disk file operations. EOT tells the computer that control should be transferred to the statement labeled ENDTAP if it runs out of tape on deck 2.

WRITE       2,NAME,ADR

This is very similar to the disk WRITE operations, but here all you need to specify is the cassette deck where the data should be written and the values that should be written there.

BKSP        2

This tells the processor to backspace the cassette in deck 2 by one record. This leaves us just enough room to write an End-of-File mark. Notice, however, that this destroys the last name and address that we wrote to cassette so we'd have room for the End-of-File mark.

WEOF        2

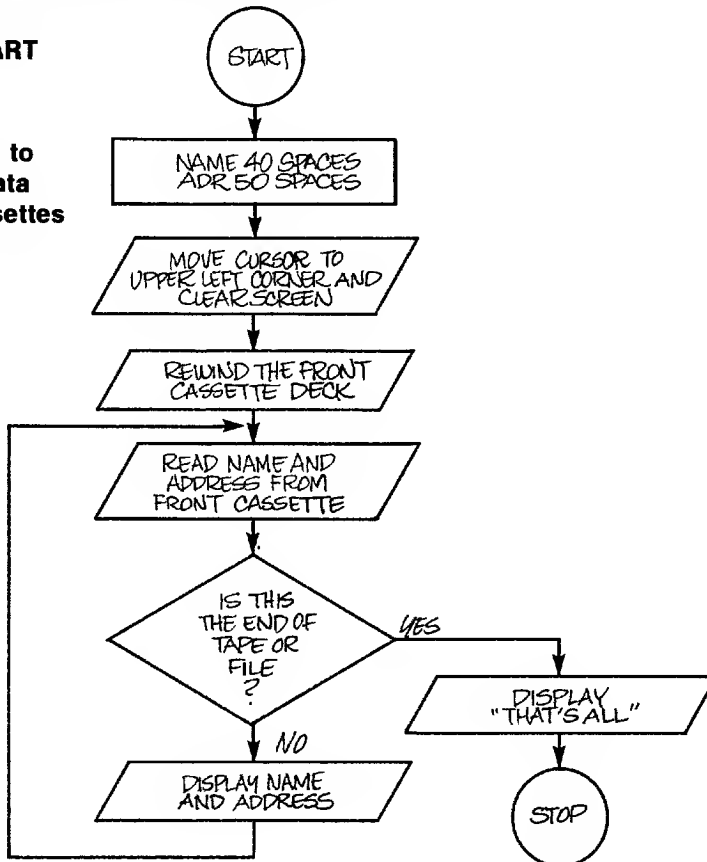
This writes an End-of-File mark on the cassette in deck 2.

### Reading from Cassettes

Now, let's write a program to read the data off the tape.

### FLOW CHART

**Program to  
Read Data  
From Cassettes**



## PROGRAM

```
NAME      DIM      40
ADR       DIM      50
          DISPLAY   *P1:1,*EF,"TAPE READER"
          REWIND    2
          TRAP      DONE IF EOT2
READER    READ      2,NAME,ADR
          GOTO      DONE IF OVER
          DISPLAY   NAME,ADR
          GOTO      READER
DONE      DISPLAY   "THAT'S ALL"
          STOP
```

That was a short program, wasn't it? Let's look at the few instructions you might not already be familiar with.

```
REWIND    2
```

What this instruction does should be obvious. It rewinds the front cassette deck. This positions the cassette tape readers to the very beginning of the data.

```
GOTO      DONE IF OVER
```

The OVER condition is set when an End-of-File mark is encountered while reading the data. So what this statement does is to transfer control to the statement labeled DONE when the program has finished reading all of the data. The EOF marker that you wrote before alerts this trap.

```
TRAP      DONE IF EOT2
```

This is the same instruction used in the writing example. We just included it in case there was some unusual problem with the cassette tape and the End-of-File mark is encountered.

```
READ      2,NAME,ADR
```

This reads the next name and address from the front cassette deck.

### How Data is Stored on Cassettes

Data storage is very simple and direct on cassette tape. Let's look at our program as it appears on the display screen as it is executed:

```

NAME AND ADDRESS PROGRAM
NAME: HARRY AARDVARK
ADDRESS: 401 LONELY LANE, NOWHERE, NY
IS THIS THE LAST ENTRY? NO
NAME: SUSIE SINGLE
ADDRESS: 2132 ZERO STREET, OURTOWN, IL
IS THIS THE LAST ENTRY? YES

```

#### DATAPOINT PROCESSOR SCREEN

And here's what our cassette data tape looks like with these data values stored on it:

BLANK TAPE	END OF FILE	2132 ZERO STREET, OURTOWN, IL	SUSIE SINGLE	G A P	401 LONELY LANE, NOWHERE, NY	HARRY AARDVARK	CLEAR LEADER
---------------	-------------------	-------------------------------------	-----------------	-------------	------------------------------------	-------------------	-----------------

The data values are stored in the exact same order as they are input! Notice that there are gaps between the groups of information. Whenever one WRITE statement ends, a gap is left to separate the information before another WRITE statement is begun.

Likewise, whenever a READ statement starts, it starts at a gap boundary. This means that with these instructions:

```

READ    1,NAME
READ    1,NAME

```

you really read two names. The address between the names is ignored because after each READ the cassette head is automatically positioned to the next inter-record gap. To use two READ statements to read one group of information, you must conclude the first READ statement with a semicolon (;) to tell the computer that you don't want it to space over to the next gap. To illustrate this, just remember that

```

READ    1,NAME,ADR

```

has the same effect as

```

READ    1,NAME;
READ    1,ADR

```

whereas

```

READ    1,NAME
READ    1,ADR

```

will actually use the next name for the address because it automatically skipped over to the next gap after executing the first READ statement.

### **Updating the File**

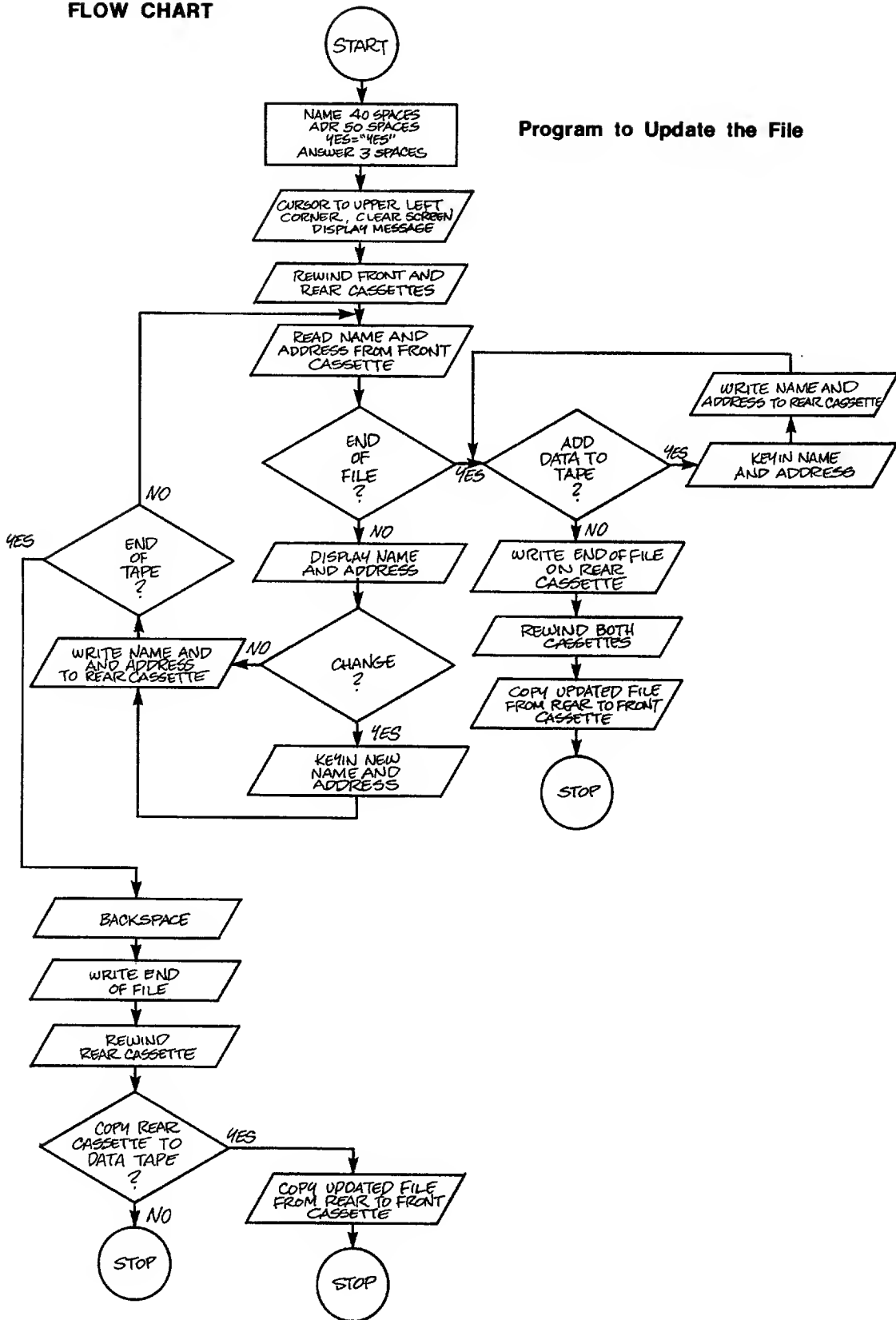
Let's assume that you saw a mistake (maybe several mistakes!) when the data file was displayed on your processor screen. Now you need to change your data file. You may want to add data to the file, also.

This program reads and displays a name and address, offers the operator the opportunity to change the values, and then writes the values out to cassette. Let's look at the flow chart:



# FLOW CHART

## Program to Update the File



Notice how we plan to read the data from the front cassette, possibly correct the data, and then write it on the rear cassette. We can even add data to the file. Then, when we're done, the rear cassette contains the updated data file. Since the rear cassette also contains the DATABUS interpreter during execution, we need to rewrite this new data to the front cassette.

Here's the program:

```

NAME      DIM      40
ADR       DIM      50
YES       INIT     "YES"
ANSWER    DIM      3
          DISPLAY  *P1:1,*EF,"CORRECTION PROGRAM"
          REWIND   1
          REWIND   2
          TRAP     EOTAPE IF EOT1
LOOP1     READ     2,NAME,ADR
          GOTO     EOFILE IF OVER
          DISPLAY  NAME,ADR
          KEYIN    "DO YOU WANT TO CHANGE IT?",ANSWER
          MATCH    ANSWER TO YES
          GOTO     CHANGE IF EQUAL
          WRITE    1,NAME,ADR
          GOTO     LOOP1
CHANGE    CALL     KEY
          WRITE    1,NAME,ADR
          GOTO     LOOP1
EOFILE    KEYIN    "ADD DATA?",ANSWER
          MATCH    ANSWER TO YES
          GOTO     ADD IF EQUAL
          WEOF     1
          REWIND   1
          PREPARE  2
LOOP2     READ     1,NAME,ADR
          GOTO     EOLOOP IF OVER
          WRITE    2,NAME,ADR
          GOTO     LOOP2
EOLOOP    WEOF     2
          STOP
ADD       CALL     KEY
          WRITE    1,NAME,ADR
          GOTO     EOFILE
EOTAPE    DISPLAY  "END OF TAPE-END OF FILE"
          BKSP     1
          WEOF     1
          REWIND   1
          KEYIN    "REWRITE DATA TAPE?",ANSWER
          MATCH    ANSWER TO YES
          GOTO     REWRITE IF EQUAL
          STOP
REWRITE   PREPARE  2

```

	GOTO	LOOP2
KEY	KEYIN	"NAME: ",NAME
	KEYIN	"ADDRESS: ",ADR
	RETURN	

The flow chart looked long and involved, but the program actually was quite direct and simple. The major new thing we've introduced here is the use of the rear cassette.

During execution, the interpreter cassette is in the rear cassette deck (deck 1). But there still is room on it for a data file. The data we write on the rear cassette won't overwrite the interpreter -- the interpreter won't let it, as explained before. Instead the data is written following the last file on the tape. Then it is rewritten to the front cassette.

Notice also how we used a subroutine, KEY, to avoid repeating instructions in our program.

### **Part 3 - Running the Program**

There are three steps you will have to go through to get your program running:

1. Type in and edit your program.
2. Compile your program. The compiler takes your instructions and translates them into a machine readable code.
3. Run your program. The interpreter handles your program's execution.

To perform these three steps, you'll need the following cassette tapes:

1. A tape with the GEDIT program on it. GEDIT is Datapoint's general purpose editor, and it lets you type in your program instructions.
2. DB2PGS 5.1, model code number 20248. This cassette contains the DATABUS 2 compiler.
3. DB2SYS 5.1, model code number 20249. This cassette contains the DATABUS 2 interpreter.
4. One extra cassette. We'll refer to this as the data tape. Use a blank cassette or an old cassette with information you don't need anymore.

To learn how to juggle these four cassettes into the two cassette drives on your processor, keep reading.

### **Step 1 -- Type In and Edit Your Program**

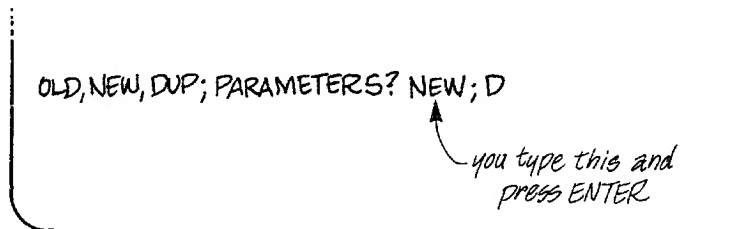
With your processor turned on and ready to go and your program written out on paper, you're ready to start entering your program.

Put the GEDIT tape in the rear cassette deck and the data tape in the front cassette deck. See the Guide to Operating Datapoint Equipment if you're not familiar with Datapoint's equipment. Press

RESTART (or RESTART and RUN on a 5500).

There are two types of GEDIT cassettes. The simplest is a "load-and-go" tape -- all you have to do is press RESTART and soon the general purpose editor will start running. The second type is a CTOS tape. If it is a CTOS tape, it is just a little more complicated, but using the RESTART Key is the place to start. In a little while "CTOS" and a version number will appear near the top of the screen in the middle of the line. "READY" will appear near the lower left corner with a flashing cursor (little rectangle of light) under it. To make sure GEDIT is on the cassette, type "CAT GEDIT" and press ENTER. If the processor responds with GEDIT/CMD, you'll know it is indeed on your cassette. Now, type "RUN GEDIT" and press ENTER. This will start running the GEDIT program.

Whether you asked CTOS to run GEDIT or you have a load-and-go GEDIT, you will soon know you are in the GEDIT program because you will get the message "OLD,NEW,DUP;PARAMETERS?" displayed, as shown in the following picture.



OLD,NEW,DUP;PARAMETERS? NEW;D

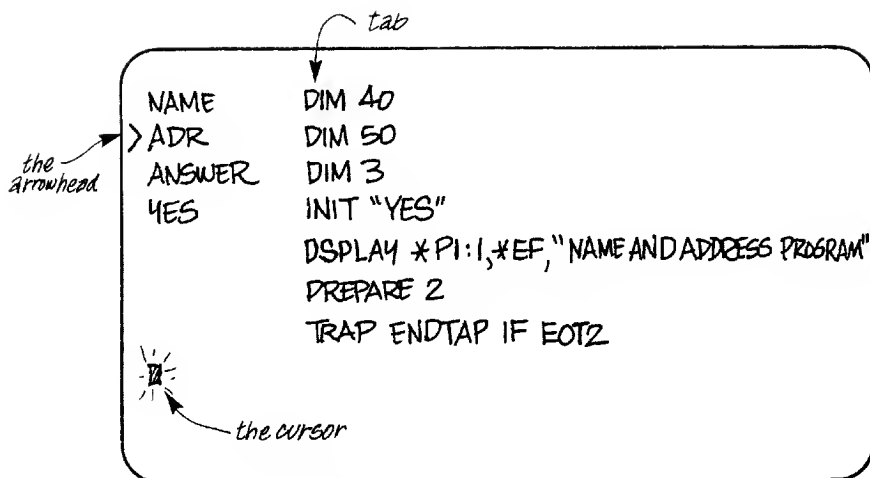
*you type this and  
press ENTER*

DATAPOINT PROCESSOR SCREEN

In response to the question, type "NEW;D" and press ENTER. "NEW" tells the editor that you want to create a new file. ";D" tells the editor that you are going to type in a DATABUS program and you have a data tape in the front cassette deck.

Incidentally, from now on we're going to assume that you know to press ENTER after each line you type in. Until you press ENTER, though, you can correct characters in the line by using the BACKSPACE and CANCEL Keys.

Soon the screen will clear, an arrowhead will appear on the left, and the cursor will blink on the left end of the bottom line. Start typing in your program. Notice how you can see what you've typed on the screen. And notice how a tab is set around column 9. Whenever you hit the space bar before that column, the cursor automatically jumps the necessary number of spaces to column 9. Look at our program:



Look closely -- we spelled DISPLAY wrong. How are we going to correct it? The editor has special commands that let us fix up mistakes like this.

All of the editor commands are prefaced with a colon (:) so the editor can tell a command from a normal line of text. These are some common editor commands:

:DEL	Delete. Erases the entire line and lets you try again.
:INS	Insert. Opens up a space between two lines so an extra line may be squeezed in.
:MOD	Modify. Lets you change individual characters or a group of characters. In our case, we typed DSPLAY instead of DISPLAY. To correct it we type:

```
:MOD DSPLAY<DISPLAY
```

The line will now contain the proper spelling.

```
DSPLAY<DISPLAY
```

```

      ↑   ↑   ↑
  (old) (replaces) (new)

```

One last step -- when you've got everything so it looks good, stop. Contemplate your work of art. Then type in the last and most important editor command.

:END	End of Program. Indicates to the editor that you are done and to write a complete, perfect copy of his program to the front tape.
------	---

Wait until either the CTOS message or "OLD,NEW,DUP,PARAMETERS?" reappears, then go on to the next step.

## Step 2 -- Compiling Your Program

Leave the data tape that now contains your program in the front cassette deck. Replace the GEDIT tape in the rear deck with the DB2PGS tape. The purpose of this step is to let the compiler make machine readable code out of your statements.

First of all, the compiler must know what type of machine you are using. Compilers and other programs cannot make intuitive guesses -- they must be told all of the facts about their operation. Configuring is a word that is used to describe the process of telling the compiler what type of machine you are using.

If this is the first time you've ever run this compiler, the configuration questions will automatically be asked. So press RESTART and wait for the questions. If the cassette has been used before, the configuration values will not be asked. However, if you depress the KEYBOARD Key while you press ENTER, and hold it down until a message appears on the screen, you can change or check the configured values.

Your display screen will look like this:

```

D B C M P  RELEASE 5.1
CONFIGURE PROGRAM TAPE?

DATAPOINT PROCESSOR SCREEN
```

Let's go over all of the questions that are asked. Press ENTER after responding to each question.

### CONFIGURE PROGRAM TAPE?

Answer "Y" if this is the first time through or you want to change the values. Answer "N" and none of the values you type in will be recorded.

### OBJECT MACHINE SIZE (8,12,16)?

Type in the memory size of the machine you are using. If you're really unsure, and there's no readily available means to finding out for sure what the memory size is, type "8".

### PRINT?

Answer "Y" to get a copy of your program and any error messages printed.

### LOCAL OR SERVO PRINTER?

This is asked if you responded with a "Y" to the PRINT question. Type "S" for servo or "L" for any other type of printer.

## DISPLAY?

Answer "Y" to see the results of your program's compilation on the screen of your Datapoint processor.

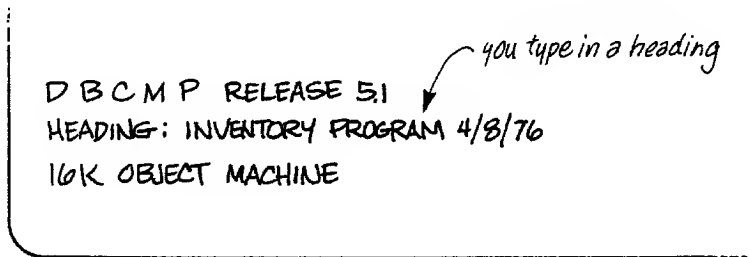
## CODE?

If you answered "Y" to the DISPLAY or PRINT questions, this question will be asked. You will usually want to answer "N" so your time isn't wasted with the printing or displaying of the machine code generated by the compiler.

There's one last question that will be asked every time you use the compiler if a print out is desired:

## HEADING?

Obviously, you should type in an appropriate heading here. You might want to include the date and time for future reference.



```
D B C M P RELEASE 5.1
HEADING: INVENTORY PROGRAM 4/8/76
16K OBJECT MACHINE
```

## DATAPOINT PROCESSOR SCREEN

As soon as you press the ENTER Key after typing in the heading, the compiler will start working. Sit back and relax while the compiler is busy. You'll know that it is done when DONE and DB2CMP 5.1 message reappears. The data cassette in the front deck now contains an extra file with the compiled version of machine code in it.

If you have other programs to compile, now is the time to switch front cassettes and compile them. Note that the compiler now is configured, and you won't be asked any of the questions except, HEADING if a print out is desired.

### Compiler Error Messages

The following errors can occur during compilation:

D                      The D flag means DOUBLE DEFINITION. It is flagged if a label or variable has been defined to more than one value during compilation.

I                      The I flag means INSTRUCTION UNKNOWN. This means that the instruction you are using is not understood by the compiler. Maybe you spelled it wrong?

E                               The E flag means that there has been an ERROR in the statement. This can be caused by any unrecognizable character.

U                               The U flag means UNDEFINED SYMBOL. It is used whenever a label is referenced but not defined.

OVERFLOW                    This message is given if your program is too large. Try dividing your program into two or more shorter programs.

DICTIONARY FULL            This message is given if your program is using too many labels and variables. Try dividing your program into two or more shorter programs.

What do these error messages look like? Let's take a look at our cassette data file writing program print out.

17000	NAME	DIM	40
17053	ADR	DIM	50
17140	ANSWER	DIM	3
17146	YES	INIT	"YES"
E17154		DISPLAY	*ES,"NAME AND ADDRESS PROGRAM"
17207		PREPARE	2
17211		TRAP	ENDTAP IF EOT2
17214	WRITER	KEYIN	"NAME: ",NAME
17225		KEYIN	"ADDRESS: ",ADR
17241		WRITE	2,NAME,ADR
17246		KEYIN	"IS THIS THE LAST ENTRY?",ANSWER
17301		MATCH	ANSWER TO YES
17304		GOTO	FINISH IF EQUAL
17307		GOTO	WRITER
17311	ENDTAP	DISPLAY	"YOU'RE OUT OF TAPE"
17335		BKSP	2
17337	FINISH	WEOF	2
17341		BEEP	
17342		STOP	
17343		STOP	
17344	ENDTAP		
17346	WRITER		
17350	FINISH		
17352	NAME		
17354	ADR		
17356	ANSWER		
17360	YES		

ERRORS WERE E



Notice how the statement with the mistake is flagged with an error. Also, at the bottom of the print out is a summary of the errors. In this case, we used \*ES in the DISPLAY statement, but cassette DATABUS doesn't recognize \*ES. You have to use \*P1:1,\*EF to erase the screen.

### Step 3 -- The Interpreter

Your program now is compiled into a form that can be read by the computer. Now you want the computer to read it and run it. For this step, you use the interpreter.

Leave the data tape in the front cassette deck and put the DB2SYS 5.1 cassette in the rear cassette deck. Press RESTART (or RESTART and RUN on a 5500).

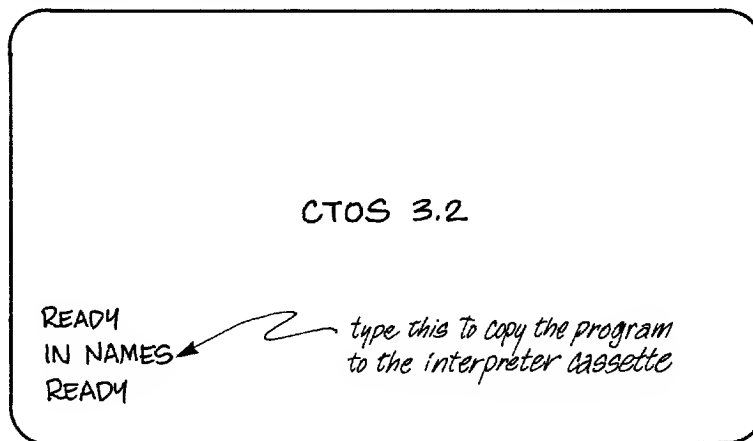
Soon the CTOS message and "READY" will appear on the screen.

The DB2SYS cassette consists of:

1. A cassette loader block
2. File 0: CTOS (the cassette tape operating system)
3. File 1 CTOS Catalog (a directory of where all of the files are on the cassette)
4. File 2 DATABUS 2 Interpreter (DB2INT)
5. File 3 DATABUS 2 Master Program (DB2MAS)

The first thing you will want to do is to add your compiled program to this interpreter cassette. First, think of a name for your program. It must begin with an alphabetic character and contain an additional one to five alphabetic or numeric characters. We'll call ours NAMES.

To copy the program on the front cassette to the rear cassette, type "IN NAMES" and press ENTER. And in just a minute or so "READY" will reappear and you should remove the front cassette -- it isn't needed anymore (unless you need to correct your program later)!



DATAPOINT PROCESSOR SCREEN

### File Numbers

NAMES is now File 4 on the DB2SYS cassette. If you wanted to add another program, ORDERS, and followed the same procedure, it would be File 5. Files are numbered in octal -- that is , from 01 to 07, 10 to 17, 20 to 27, and 30 to 37. The first 14 of these files (files 01 to

17) also have names. But the last 16 files can only be called and added by number.

What if you forget what file number your program is? Type "CAT" and you'll get a listing of the files on the cassette.

```
READY
CAT
DB2INT DB2MAS NAMES ORDERS
READY
```

DATAPoint PROCESSOR SCREEN

Notice that File 0 and File 01 are not listed. DB2INT is File 02, DB2MAS is file 03, NAMES is file 04, and ORDERS is file 05.

Think for a moment about the program you are about to execute. Does it write data to the front cassette? Then you should put a cassette in the front cassette deck. Does it read data from the front cassette? Make sure that the front cassette deck contains the tape with the data on it.

### The Interpreter

And now you're ready to run your program. As with the compiler, the first time ever that you run the interpreter you must configure it (tell the program what type of computer you are using).

To run the interpreter, type "RUN DB2INT" and press ENTER. If this is the first time, the configuration questions will be asked.

```
READY
RUN DB2INT
DATABUS 2 INTERPRETER - RELEASE 5.1
CONFIGURE PROGRAM TAPE?
```

DATAPoint PROCESSOR SCREEN

These are the configuration questions that will be asked:

CONFIGURE PROGRAM TAPE?

Answer "Y" to record your answers on the cassette.

INTERPRETER TAPE LGO OR CTOS?

Type "C" to indicate that you are using a CTOS tape.

WRITE VERIFY?

Answer "Y" if each WRITE operation should be double checked for accuracy, otherwise answer "N".

Next, the MASTER program is initialized:

```
WRITE VERIFY? N
DATABUS 2 INTERPRETER - RELEASE 5.1
UTILITY MASTER - DATABUS 2 RELEASE 5.1
READY
```

*the cursor waits for a file number to execute*

DATAPOINT PROCESSOR SCREEN

The cursor is flashing, waiting for you to type in a file number (not a name!) We want to execute NAMES, which is file 04. So we type in 04, press ENTER, and soon enough we get to see the results of our labors -- the program is actually running!

When it's done, "READY" will reappear. Take out the DB2SYS cassette if you're done with the interpreter.

Are you satisfied with your program? If not, you can delete the one you just put on the DB2SYS cassette by typing "DELETE NAMES" (use your program name, of course). If you want to delete a file other than the last one on the cassette, be sure to put an extra cassette - one you don't need the information from - in the front deck. This extra cassette is necessary so the files on the DB2SYS cassette can be rearranged.

Now use the GEDIT program to fix your program, recompile it, and then load it on the DB2SYS tape again and run it. Hopefully you'll like it better with the changes you've made.

## APPENDIX A

### Instruction Summary

#### Syntactic Definitions

condition	The result of any arithmetic or string operation: OVER, LESS, EQUAL, ZERO, or EOS (EQUAL and ZERO are two names for the same condition).
character string	Any string of printing ASCII characters.
event	The occurrence of a program trap: PARITY, RANGE, FORMAT, CFAIL, or IO.
list	A list of variables or controls appearing in an input/output instruction.
name	Any combination of letters (A-Z) and digits (0-9) starting with a letter (only the first eight characters are used).
label	A name assigned to a statement.
nvar	A name assigned to a statement defining a numeric string variable.
nval	A name assigned to an operand defining a numeric string variable or an immediate numeric value.
nlit	A constant numeric value, enclosed in double quotes(" ").
svar	A name assigned to a statement defining a character string variable.
sval	A name assigned to an operand defining a character string variable or a quoted alphanumeric character.
slit	A constant character string, enclosed in double quotes ("").
nlist	A series of contiguous numeric variables.
slist	A series of contiguous string variables.
rn	A positive record number (<=0) used to randomly READ or WRITE on a file.

seq	A negative number (<0) used to READ or WRITE on a file sequentially.
key	A non-null string used as a key to indexed accesses.
null	A null string used as a key to an indexed read.

## DATABUS 11 Language Summary

For the following summary:

Items enclosed in brackets | are optional.

Items separated by the | symbol are mutually exclusive (one or the other but not both must be used).

### COMPILER DIRECTIVES

label	EQU	10
label	EQUATE	100
	INC	filename   /ext
	INCLUDE	filename   /ext

### FILE DECLARATIONS

label	FILE
label	IFILE

### DATA DEFINITIONS

label	FORM	n.m
label	FORM	"456.23"
label	DIM	n
label	INIT	"character string"
label	FORM	*n.m
label	FORM	**"456.23"
label	DIM	*n
label	INIT	**"CHARACTER STRING"

### CONTROL

GOTO	(label)
GOTO	(label) IF (condition)
GOTO	(label) IF NOT (condition)
BRANCH	(nvar) OF (label list)
CALL	(label)
CALL	(label) IF (condition)

CALL	(label) IF NOT (condition)
RETURN	
RETURN	IF (condition)
RETURN	IF NOT (condition)
STOP	
STOP	IF (condition)
STOP	IF NOT (condition)
CHAIN	(sval)
CHAIN	(slit)
TRAP	(label) IF (event)
TRAPCLR	(event)
ROLLOUT	(svar)
ROLLOUT	(slit)

## CHARACTER STRING HANDLING

MATCH	(svar) TO (svar)
MATCH	(slit) TO (svar)
MOVE	(svar) TO (svar)
MOVE	(slit) TO (svar)
MOVE	(svar) TO (nvar)
MOVE	(nlit) TO (nvar)
MOVE	(nvar) TO (svar)
APPEND	(svar) TO (svar)
APPEND	(slit) TO (svar)
APPEND	(nvar) TO (svar)
CMOVE	(sval) TO (svar)
CMATCH	(sval) TO (sval)
BUMP	(svar)
BUMP	(svar) BY (nlit)
RESET	(svar) TO (sval)
RESET	(svar) TO (nvar)
RESET	(svar)
ENDSET	(svar)
LENSET	(svar)
CLEAR	(svar)
EXTEND	(svar)
LOAD	(svar) FROM (nvar) OF (slist)
STORE	(svar) INTO (nvar) OF (slist)
STORE	(slit) INTO (nvar) OF (slist)
CLOCK	TIME TO (svar)
CLOCK	DAY TO (svar)
CLOCK	YEAR TO (svar)
TYPE	(svar)
SEARCH	(nvar) IN (nlist) TO (nvar) WITH (nvar)
SEARCH	(svar) IN (slist) TO (nvar) USING (nvar)
REPLACE	(svar) IN (svar)
REP	(slit) IN (svar)

## ARITHMETIC

ADD	(nvar) TO (nvar)
ADD	(nlit) TO (nvar)
SUB	(nvar) FROM (nvar)
SUB	(nlit) FROM (nvar)
SUBTRACT	(nlit nvar) FROM (nvar)
MULT	(nvar) BY (nvar)
MULT	(nlit) BY (nvar)
MULTIPLY	(nlit nvar) BY (nvar)
DIV	(nvar) INTO (nvar)
DIV	(nlit) INTO (nvar)
DIVIDE	(nlit nvar) INTO (nvar)
MOVE	(nvar) TO (nvar)
MOVE	(nlit) TO (nvar)
COMPARE	(nvar) TO (nvar)
COMPARE	(nlit) TO (nvar)
LOAD	(nvar) FROM (nvar) OF (nlist)
STORE	(nvar) INTO (nvar) OF (nlist)
STORE	(nlit) INTO (nvar) OR (nlist)
CHECK11	(svar) BY (svar)
CK11	(svar) BY (slit)
CHECK10	(svar) BY (svar)
CK10	(svar) BY (slit)

## INPUT/OUTPUT

KEYIN	(list)
DISPLAY	(list)
BEEP	
PRINT	(list)
PREPARE	(file),(svar  slit)
PREP	(file),(svar  slit)
OPEN	(file  ifile),(svar  slit)
CLOSE	(file  ifile)
WRITE	(file  ifile),rn  seq  key  (list)
WRITAB	(file),rn  seq;(list)
WEOF	(file  ifile),rn  seq
UPDATE	(ifile)      (list)
READ	(file  ifile),rn  seq  key  nul;(    (list)      )
READKS	(ifile);(    (list)      )
DELETE	(ifile),(svar)
INSERT	(ifile),(svar)

## DATABUS 2 Language Summary

NOTE: The following two syntactic definitions apply to the DATABUS 2 language:

1. Conditions are OVER, LESS, EQUAL, ZERO, and EOS.
2. Events are EOT1, EOT2, RFAIL1, RFAIL2, FORM1,

## DIRECTIVES

FORM	n.m
FORM	"456.23"
DIM	n
INIT	"character string"
FORM	*n.m
FORM	*"456.23"
DIM	*n
INIT	*"character string"

## CONTROL

TRAP	(label) IF (event)
TRAPCLR	(event)
GOTO	(label)
GOTO	(label) IF (condition)
GOTO	(label) IF NOT (condition)
CALL	(label)
CALL	(label) IF (condition)
CALL	(label) IF NOT (condition)
RETURN	
RETURN	IF (condition)
RETURN	IF NOT (condition)
STOP	
STOP	IF (condition)
STOP	IF NOT (condition)
CHAIN	(svar)
BRANCH	(nvar) OF (label list)

## STRING

CMATCH	(sval) TO (sval)
CMOVE	(sval) TO (svar)
MATCH	(svar) TO (svar)
MOVE	(svar) TO (svar)
MOVE	(svar) TO (nvar)
MOVE	(nvar) TO (svar)
APPEND	(svar) TO (svar)
RESET	(svar) TO (sval)
RESET	(svar) TO (nvar)
RESET	(svar)
BUMP	(svar) BY (literal)
BUMP	(svar)
ENDSET	(svar)
LENSSET	(svar)
TYPE	(svar)
EXTEND	(svar)
CLEAR	(svar)



LOAD (svar) FROM (nvar) OF (svar list)  
STORE (svar) INTO (nvar) OF (svar list)

#### NUMERIC VARIABLE ARITHMETIC

ADD (nvar) TO (nvar)  
SUB (nvar) FROM (nvar)  
MULT (nvar) BY (nvar)  
DIV (nvar) INTO (nvar)  
MOVE (nvar) TO (nvar)  
COMPARE (nvar) TO (nvar)  
LOAD (nvar) FROM (nvar) OF (nvar list)  
STORE (nvar) INTO (nvar) OF (nvar list)

#### KEYBOARD, C.R.T., PRINTER I/O

KEYIN (list)  
KEYIN (list);  
DISPLAY (list)  
DISPLAY (list);  
PRINT (list)  
PRINT (list);  
BEEP  
CLICK  
DSENSE  
KSENSE

#### CASSETTE TAPE I/O

READ (unit),(list)  
READ (unit),(list);  
READ (unit)  
WRITE (unit),(list)  
REWIND (unit)  
BKSP (unit)  
PREPARE (unit)  
WEOF (unit)

## APPENDIX B

### Editor Commands

The DOS Editor program enables you to create and modify files. All Editor commands are prefixed with a colon (:) to distinguish them from text lines. The pointer must be positioned at the line that needs correcting (use the KEYBOARD and DISPLAY Keys to do this).

A full description of all Editor commands is in the DOS User's Guide. Here is an abbreviated list:

:D	Delete entire line.
:D text	Delete all characters from the left edge of the line through and including the specified text.
:E	End of file. Write file.
:E*	Display last line of file on screen.
:EO	Display data continuously on screen through last line of file.
:F text	Find line starting with text.
:I	Insert a line.
:L	Show next line in the file.
:L text	Find imbedded text.
:M old<new	Replace old text with new text.
:SC	Remove the lines from top of screen down to and including the pointed line.
:SB	Remove the lines through the bottom of the screen.



## NOTES

# DATAPoint CORPORATION



**The Leader in Dispersed Data Processing™**

9725 Datapoint Drive  
San Antonio, Texas 78284  
512/690-7151